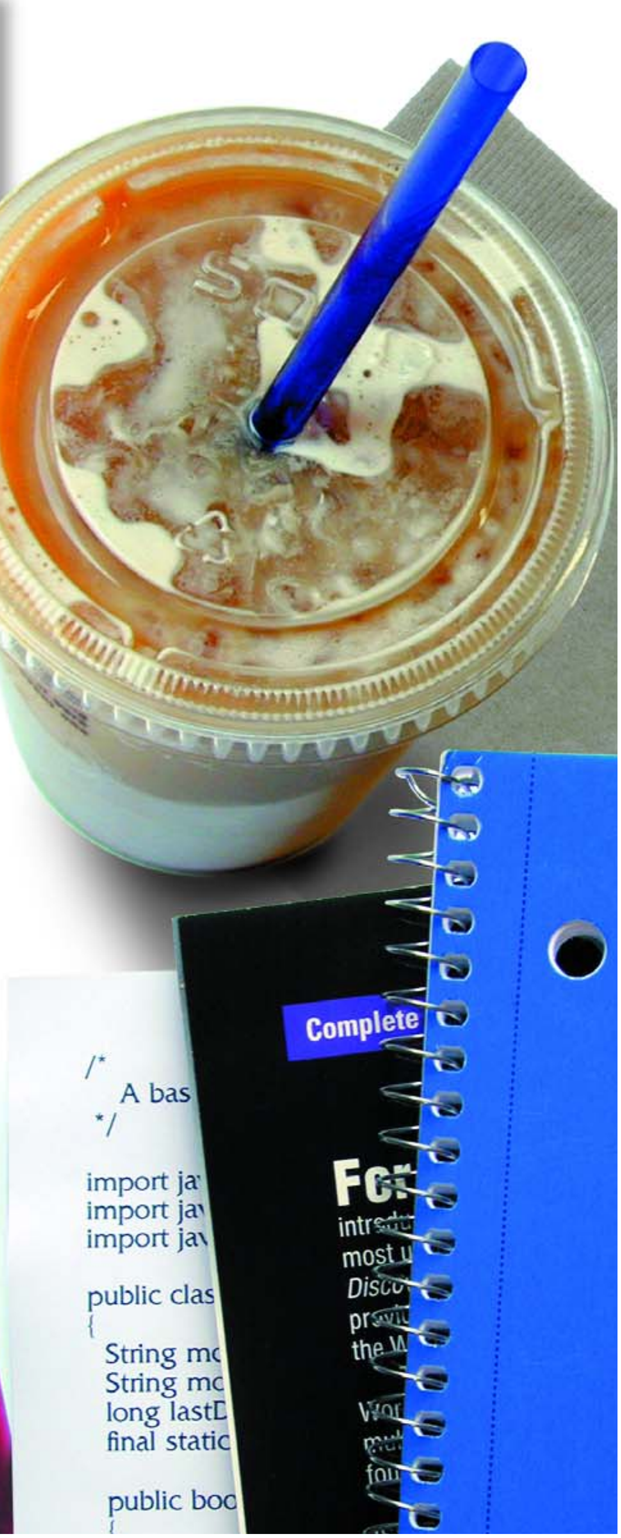
CHAPTER 15

Program Development and Programming Languages

What a relief! After a day of midterms, your last exam finally is finished. Earlier today, all of your friends completed their tests, and they are waiting for you at Java Joint, a local coffee house. With all the homework and studies this semester, fun and relaxing times with friends are rare. Your excitement builds.

When you join the group, they are in the middle of a conversation about Java. You decide to order the house specialty, vanilla iced latte. It hits the spot! Now you can focus on the group's discussion. Yes, you agree this new Java is great. Your friends describe it as simple and robust, but you think of it more as savory and delectable.

Java applet, you hear one friend mention. Instantly you realize it has happened again. Your computer-major friends have not been talking about the house special at all. They are talking about a Web programming language. As you sit back in your chair with your latte, you hope no one will notice the dialogue is way over your head. You resolve to surf the Internet tomorrow to learn more about Java — the programming language.



OBJECTIVES

After completing this chapter, you will be able to:

- Explain the six steps in the program development life cycle
- Describe top-down program design
- Explain structured program design and the three basic control structures
- Explain the differences among the categories of programming languages
- Describe the object-oriented approach to progam development
- Identify programming languages commonly used today
- Identify the uses of application generators, macros, and RAD tools
- Describe various Web page development tools, including HTML, DHTML, XML, and WML
- Identify uses of multimedia authoring packages

WHAT IS A COMPUTER PROGRAM?

A **computer program** is a set of instructions that directs a computer to perform tasks. Computer programmers use program development tools that generate these instructions automatically or use a programming language to write these instructions themselves. A programming language is a set of words, symbols, and codes that enables a programmer to communicate instructions to a computer.

It is possible you will use a program development tool. Although you may never write a computer program, information you request may require that a programmer modify or write a program. Thus, you should understand how programmers develop programs to meet information requirements.

THE PROGRAM DEVELOPMENT LIFE CYCLE

The **program development life cycle** (**PDLC**) is a set of steps programmers use to build computer programs. As presented in Chapter 14, the SDLC guides information technology (IT) professionals through the development of an information system. Likewise, the PDLC guides computer programmers through the development of a program. The PDLC consists of six steps (Figure 15-1):

1. Analyze Problem
2. Design Programs
3. Code Programs
4. Test Programs
5. Formalize Solution
6. Maintain Programs

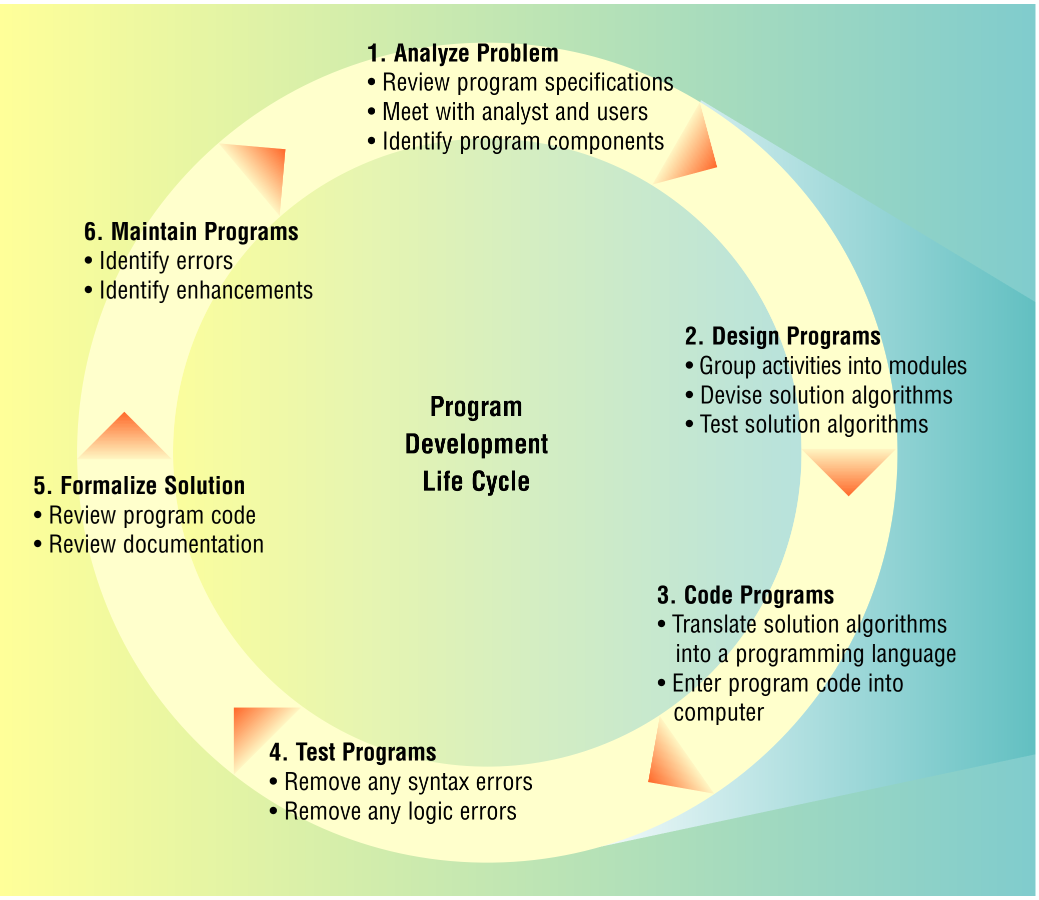

1. Analyze Problem
• Review program specifications
• Meet with analyst and users
• Identify program components
6. Maintain Programs
• Identify errors
• Identify enhancements
2. Design Programs
• Group activities into modules
• Devise solution algorithms
• Test solution algorithms
Program Development Life Cycle
5. Formalize Solution
• Review program code
• Review documentation
3. Code Programs
• Translate solution algorithms into a programming language
• Enter program code into computer
4. Test Programs
• Remove any syntax errors
• Remove any logic errors

Figure 15-1 The program development life cycle consists of six steps that form a loop.

As shown in Figure 15-1, the steps in the PDLC form a loop. Program development is an ongoing process within system development. The Maintain Programs step connects to the Analyze Problem step to form the loop. Each time someone identifies errors in or improvements to a program and requests program modifications, the Analyze Problem step begins again.

What Initiates the Program Development Life Cycle?

As discussed in Chapter 14, the system development life cycle (SDLC) consists of five phases: planning, analysis, design, implementation, and support. During the analysis phase, the development team recommends how to handle software needs. Choices include building custom software, purchasing packaged software, or outsourcing the entire IT operation.

If the company decides to build custom software, it then faces additional decisions. Should it develop the software in-house using its own IT staff? Should it contract the services of an outside source to develop the software? Or, should it consider some combination of these two alternatives?

If the company opts for in-house development, the design and implementation phases of the SDLC become quite extensive. In the design phase, the analyst creates a detailed set of requirements for the programmers. These design specifications, called the **program specification package**, identify the input, output, processing, and data requirements of each program and the relationships among programs. Once the programmers receive the program specification package, the implementation phase begins. At this time, the programmer analyzes the problem to be solved. The PDLC thus begins at the start of the implementation phase (Figure 15-2).

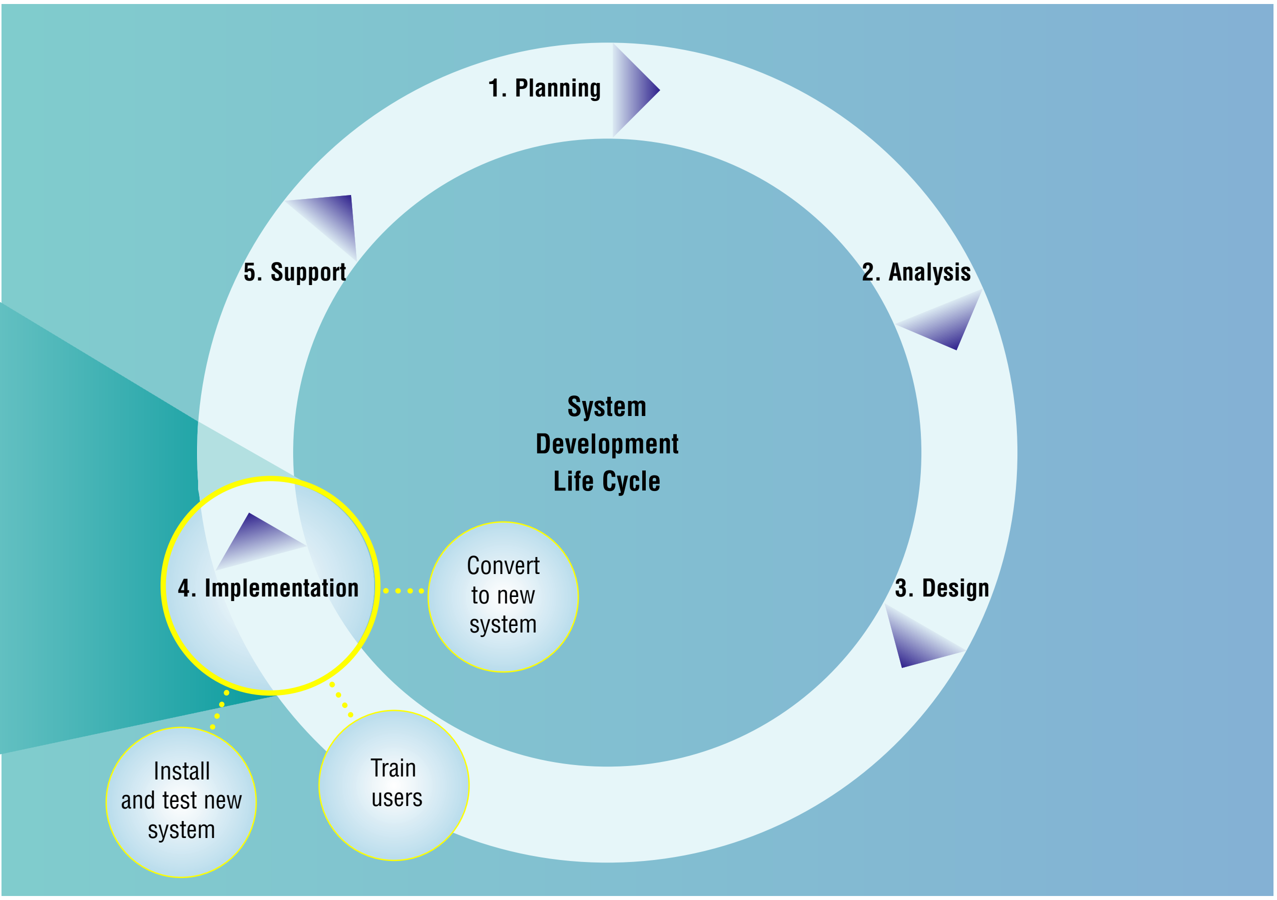


Figure 15-2 The program development life cycle is part of the implementation phase of the system development life cycle.

The scope of the program specification package largely determines how many programmers work on the program development. If scope is large, a **programming team** that consists of a group of programmers may develop the programs. If the specifications are fairly simple, a single programmer might complete all of the development tasks. Whether a single programmer or a programming team, all of the programmers involved must interact with users and members of the development team throughout the PDLC.

By following the steps in the PDLC, programmers can create programs that are correct (produce accurate information) and maintainable (easy to modify). Interacting with the development team further helps programmers build programs to meet user requirements.

The following sections address each of the steps in the PDLC.

STEP 1 – ANALYZE PROBLEM

The first step in the PDLC is to analyze the problem the program(s) should solve so you can begin to develop an appropriate solution. In most cases, the solution requires more than one program. This analysis step consists of three major tasks: (1) review the program specifications package, (2) meet with the systems analyst and users, and (3) identify each program's input, output, and processing components.

First, the programmer reviews the program specifications package. This package identifies the input, output, processing, and data requirements for each program. It also shows relationships among programs.

The program specifications package contains a variety of deliverables such as charts, diagrams, and reports. These deliverables document various aspects of the users' requirements. For example, screen and report layout charts show input and output requirements. System flowcharts, structured English, decision tables, and decision trees convey program design (processing) requirements. The data dictionary identifies the data requirements. By thoroughly reviewing these deliverables, the programmer understands the nature and requirements of each program.

During this step, the programmer also meets with the systems analyst and the users. This enables the programmer to understand the purpose of the program from the users' perspective. Recall from Chapter 14 that a guideline of system development is to involve users throughout the entire SDLC.

After reviewing the program specifications package and meeting with the systems analyst and users, the programmers may wish to recommend a change to some aspect of the program. In this case, they discuss the change with the systems analyst and the users. If everyone agrees, the programmer modifies the design specifications. A programmer never should make any change without both the systems analyst's and users' approval.

Once design specifications are established, the programmer defines the input, processing, and output (IPO) requirements for each program. Many programmers use an IPO chart for this task (Figure 15-3). An **IPO chart**, also called a **defining diagram**, identifies a program's inputs, its outputs, and the processing steps required to transform the inputs into the outputs. Programmers should review the contents of the IPO chart with the systems analyst and the users. This allows programmers to be sure they completely understand the purpose of the program. The next step is to design the programs that will meet these requirements.

APPLY IT!

Programmers' Code of Ethics

If your career goal is to be a programmer, you should consider the advantages of belonging to a professional organization. The International Programmer Guild (IPG) is a professional organization designed to represent and assist programmers. Many professionals such as accountants, doctors, lawyers, teachers, and so on are required to demonstrate their professional competence through certification and licensing. The IPG provides several certification paths and encourages members to be certified as a way to demonstrate their professional competence. The certification paths are as follows, with each level representing an increase in programming knowledge and skills:

- Junior Member
- Interim Member
- Fellow Programmer
- Senior Programmer
- Master Programmer

Membership in the IPG requires adherence to the Guild's Professional Code of Ethics, which outlines the principles and guidelines for professionalism in the programming field. This code addresses the duties of a professional programmer to the public, to their employer, in independent practice to a client, to other professional programmers, and to himself or herself.

For more information on the IPG, visit the Discovering Computers 2002 Apply It Web page (**scsite.com/dc2002/apply.htm**) and click Chapter 15 Apply It #1.

STEP 2 – DESIGN PROGRAMS

Designing the programs involves three tasks: (1) grouping each program's activities into modules, (2) devising a solution algorithm for each module, and (3) testing the solution algorithms. The first task is called top-down design. The last two tasks are part of a process called structured design.

Top-down design continues to focus on *what* the program should do (the requirements). Structured design determines *how* to build the programs based on the requirements. The following sections present details about top-down and structured design.

Top-Down Design

Top-down design breaks down the original set of program specifications into smaller, more manageable sections. It is easier to design smaller sections one at a time than to design the entire program at once. With top-sections. It is easier to design smaller sections one at a time than to design the entire program at once. With top-down design, the programmer uses a telescopic approach to view a program. The programmer begins with the big picture and then zooms in on the details.

The first step in top-down design is to identify the major function of a program, sometimes called the main routine. Next, you decompose (break down) the main routine into smaller sections. These smaller sections often are called subroutines because they are subordinate to the main routine. Then, analyze each subroutine to determine if it can be decomposed further. You continue decomposing subroutines until each one performs a single function. A section of a program that performs a single function is a **module**. Each subroutine also is a module. The main routine often is called the main module.

INPUT	PROCESSING	OUTPUT
Regular Time Hours Worked Overtime Hours Worked Hourly Pay Rate	Read regular time hours worked, overtime hours worked, hourly pay rate. Calculate regular time pay. If employee worked overtime, calculate overtime pay. Calculate gross pay. Print gross pay.	Gross Pay

Figure 15-3 An IPO (Input Process Output) chart is a tool that assists the programmer in analyzing a program.

Web Link

For more information on top-down design, visit the Discovering Computers 2002 Chapter 15 WEB LINK page (**scsite.com/dc2002/ch15/weblink.htm**) and click Top-Down Design.

ISSUE

Calling All High School Students

Recruiting Programmers

The U.S. Department of Commerce predicts that there is a need for more than a million new technology workers. Computer scientists, computer engineers, systems analysts, programmers, and network technicians are just some of the high-demand skill sets needed. When it comes to applicants, supply is not keeping up with demand. To ease the shortage, companies are recruiting high-school students by offering them attractive training programs that lead to lucrative jobs. Will students who begin technological training right out of high school have a background general enough for other pursuits if their interest fades or the market withers? Do you think companies should be allowed to sign high-school students? Why or why not? What can be done to address long-term concerns?

For more information on IT workers in demand, visit the Discovering Computers 2002 Issues Web page (**scsite.com/dc2002/issues.htm**) and click Chapter 15 Issue #1.

Web Link

For more information on structured design, visit the Discovering Computers 2002 Chapter 15 WEB LINK page (**scsite.com/dc2002/ch15/weblink.htm**) and click Structured Design.

Programmers use a **hierarchy chart**, also called a **structure chart** or **top-down chart**, to show program modules graphically (Figure 15-4). A hierarchy chart contains rectangles and lines. The rectangles are the modules. The main module is at the top of the chart. You place all other modules below the main module, connecting modules with lines to indicate their relationships. In Figure 15-4, for example, the initialization, process, and wrap-up modules are subordinate to the main module.

Programs developed using the top-down approach benefit from the simplicity of their design. They usually are reliable and easy to read and maintain. For these reasons, many IT professionals and programmers recommend the top-down approach to program design.

Structured Design

Using top-down design, the programmer identified the modules for a program (the *what*). The next step is to use structured design to identify the logical order of the tasks required to accomplish the function of each module (the *how*).

Structured design is a technique that builds all program logic from a combination of three control structures. A **control structure**, also known as a **construct**, is a design that determines the logical order of program instructions. Each module in a program typically contains more than one control structure. Structured design uses three basic control structures: sequence, selection, and repetition.

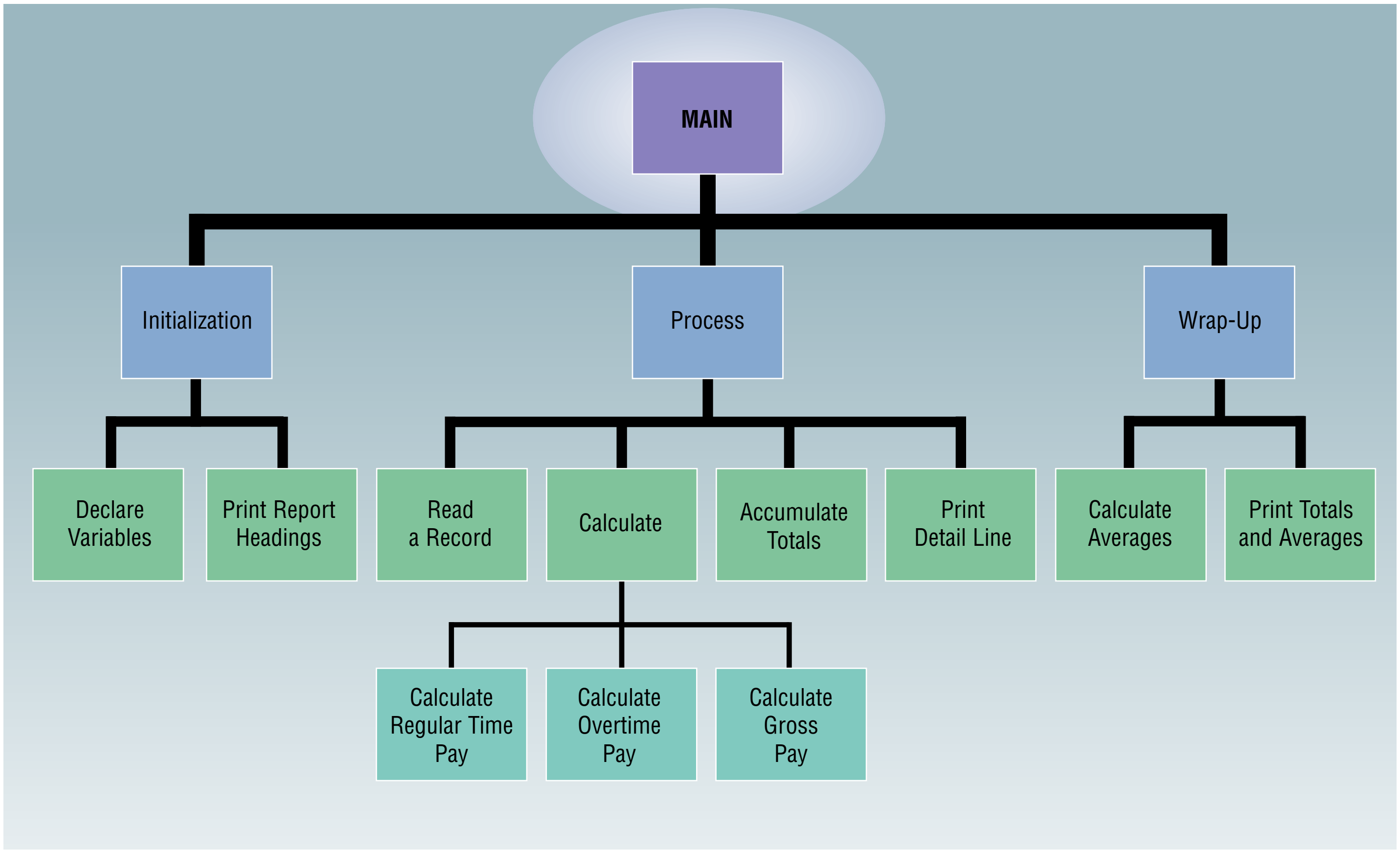


Figure 15-4 The hierarchy chart is a tool the programmer uses during top-down design. On the hierarchy chart, the program modules are drawn as rectangles. All modules are subordinate to the main module.

SEQUENCE CONTROL STRUCTURE

A **sequence control structure** shows one or more actions following each other in order (Figure 15-5). Actions include inputs, processes, and outputs. Examples of actions are reading a record, calculating averages or totals, and printing totals.

SELECTION CONTROL STRUCTURE

A **selection control structure** tells the program which action to take, based on a certain condition. Two common types of selection control structures are the if-then-else and the case.

When a program evaluates the condition in an **if-then-else control structure**, it yields one of two possibilities: true or false. Figure 15-6 shows the condition as a diamond symbol. If the result of the condition is true, then the program performs one action. If the result is false, the program performs a different (or possibly no) action. For example, the selection control structure can determine if an employee should receive overtime pay. A possible condition might be the following: Is Overtime Hours greater than 0? If the response is yes (true), then the action would calculate overtime pay. If the response is no (false), then the action would set overtime pay equal to zero.

With the **case control structure**, a condition can yield one of three or more possibilities (Figure 15-7). The size of a soft drink, for example, might be one of these options: small, medium, large, or extra large. A case control structure would determine the price of the soft drink based on the size purchased.

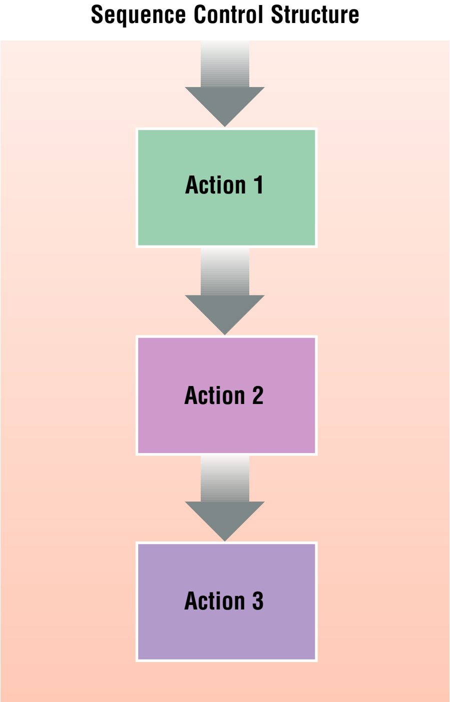


Figure 15-5 The sequence control structure shows one or more actions followed by another.

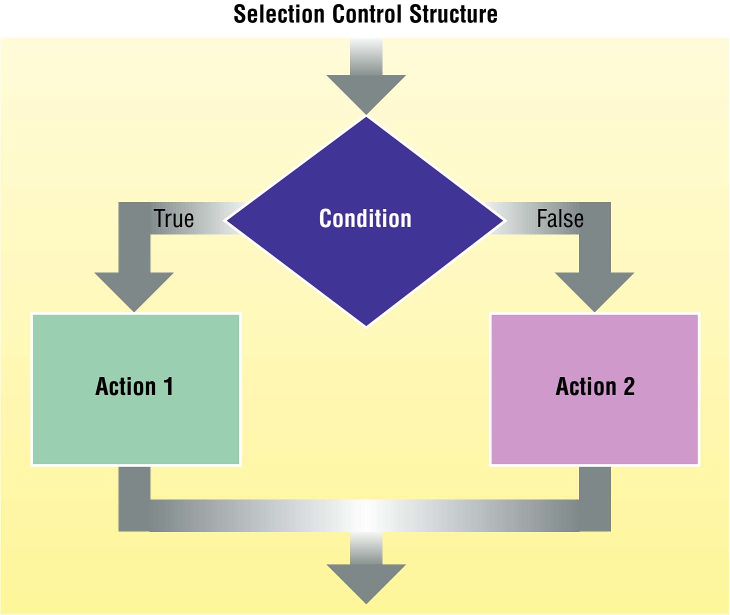


Figure 15-6 The if-then-else control structure directs the program toward one course of action or another based on the evaluation of a condition.

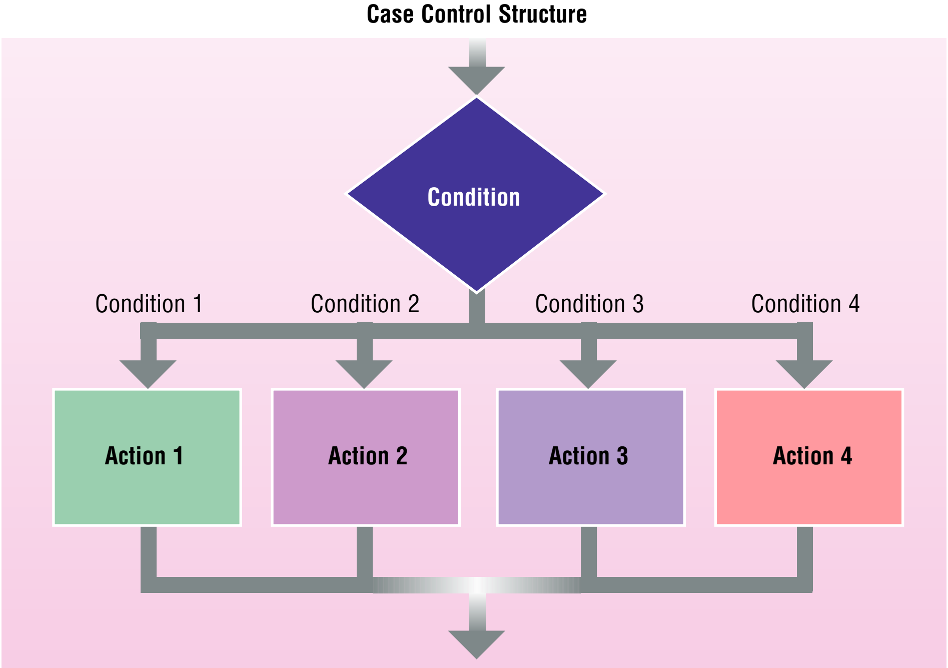


Figure 15-7 The case control structure allows for more than two alternatives when a condition is evaluated.

REPETITION CONTROL STRUCTURE

Use the **repetition control structure**, also called the **iteration control structure**, when a program performs one or more actions repeatedly as long as a certain condition is met. Many programmers refer to this construct as a **loop**. Two forms of the repetition control structure are the do-while and do-until.

The **do-while control structure** repeats one or more times as long as a condition is true (Figure 15-8). This control structure tests a condition at the beginning of the loop. If the result of the condition is true, the program executes the action(s) inside the loop. Then, the program loops back and tests the condition again. If the result of the condition still is true, the program executes the action(s) inside the loop again. This looping process continues until the condition being tested becomes false. At that time, the program stops looping and moves to another set of actions.

Do-While Control Structure

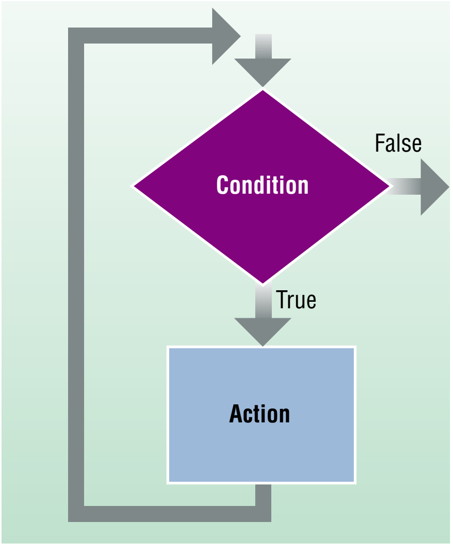


Figure 15-8 The do-while control structure tests the condition at the beginning of the loop. It exits the loop when the result of the condition is false.

Programmers frequently use the do-while control structure to process all records in a file. A payroll program using a do-while control structure, for example, loops once for each employee. This program stops looping when it processes the last employee's record.

The **do-until control structure** is similar to the do-while but has two major differences: where it tests the condition and when it stops looping. First, the do-until control structure tests the condition at the end of the loop (Figure 15-9). The action(s) in a do-until control structure thus always will execute at least once. The loop in do-while control structure, by contrast, might not execute at all. That is, if the condition immediately is false, the action or actions in the do-while loop never execute. Second, a do-until control structure continues looping until the condition is true — and then stops. This is different from the do-while control structure, which continues to loop while the condition is true.

Do-Until Control Structure

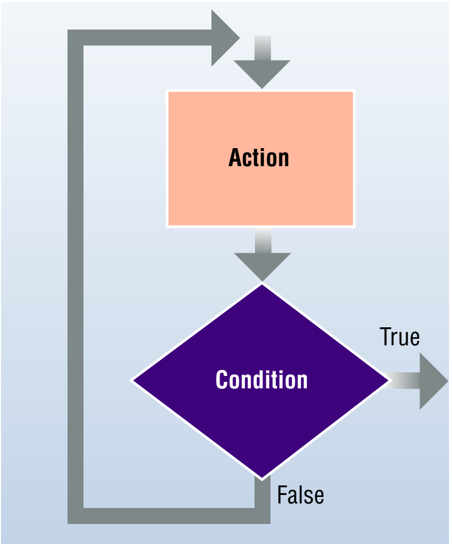


Figure 15-9 The do-until control structure tests the condition at the end of the loop. It exits the loop when the result of the condition is true.

Proper Program Design

Programs designed using top-down and structured techniques are simple, yet effective. These programs also are reliable and easy to use and maintain. When using top-down and structured techniques, a programmer must be sure to follow the guidelines of a proper program. To write a **proper program**, the programmer must make certain that the program, each of its modules, and each of its control structures has the following characteristics:

1. No dead code
2. No infinite loops
3. One entry point
4. One exit point

Dead code is any code, or program instruction, that a program never executes. When programmers write a program, they write a section of code in the program. Sometimes they decide not to use the code, but leave it in the program anyway. This unused code is known as dead code. Dead code serves no purpose and should not exist.

An **infinite loop** is a set of instructions that repeats continuously. Properly designed business programs should not contain infinite loops.

An **entry point** is the location where a program, a module, or a control structure begins. An **exit point** is the location where it ends. Figure 15-10 shows the entry and exit points of a module with two control structures: an if-then-else control structure within a do-while control structure. The entry point of the do-while control structure is just prior to the first condition. The exit point occurs when the result of this condition is false. The entry point of the if-then-else control structure occurs just prior to the second condition. The exit point occurs just after the program executes one of the two actions.

A properly designed program, module, or control structure has only one entry point and one exit point. Prior to the introduction of this concept, programmers often designed programs with multiple entry and exit points. This caused the program to jump frequently from one section of code to another. Some people use the term, spaghetti code, to refer to these poorly designed programs. This is because if you draw a line connecting all of the jumps together, the resulting line would look like cooked spaghetti! If you restrict program logic to the three basic control structures, your programs naturally will follow the single entry and single exit point rule.

Design Tools

A **solution algorithm,** also called **program logic**, is a graphical or written description of the step-by-step procedures in a module. Determining the logic for a program usually is a programmer's most challenging task. It requires that the programmer understand structured design concepts, as well as creativity. Defining the solution algorithm is both a skill and an art.

To help develop a solution algorithm, programmers use **design tools**. Three design tools are program flowcharts, Nassi-Schneiderman charts, and pseudocode.

PROGRAM FLOWCHART A **program flowchart**, or simply **flowchart**, graphically shows the logic in a solution algorithm. The American National Standards Institute (ANSI) published a set of standards for program flowcharts in the early 1960s. These standards, still used today, specify symbols for various operations in a program's logic (Figure 15-11).

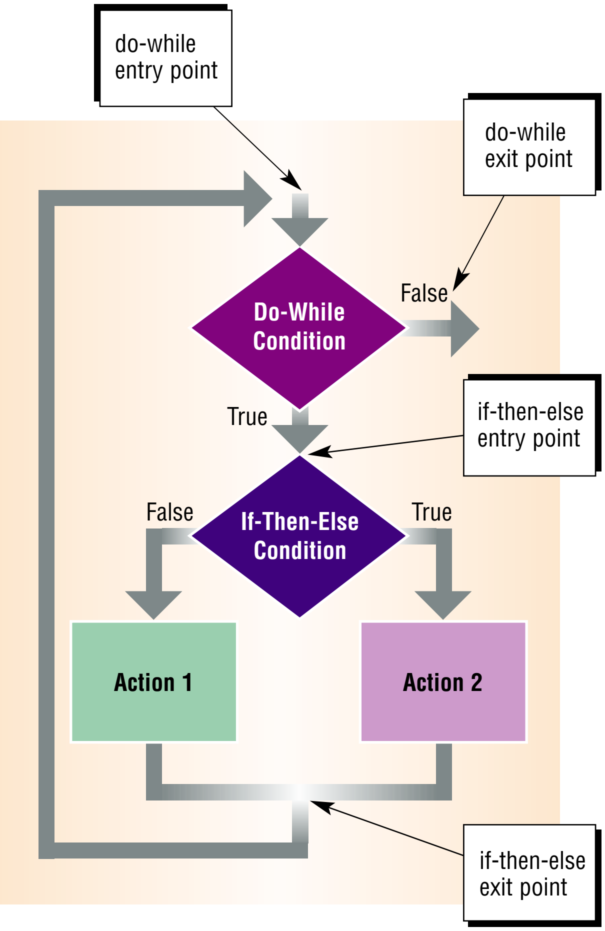


Figure 15-10 Program modules often have control structures nested inside one another, each of which should have one entry point and one exit point. In this example, an if-then-else is nested inside a do-while. The entry and exit points adhere to proper program design.

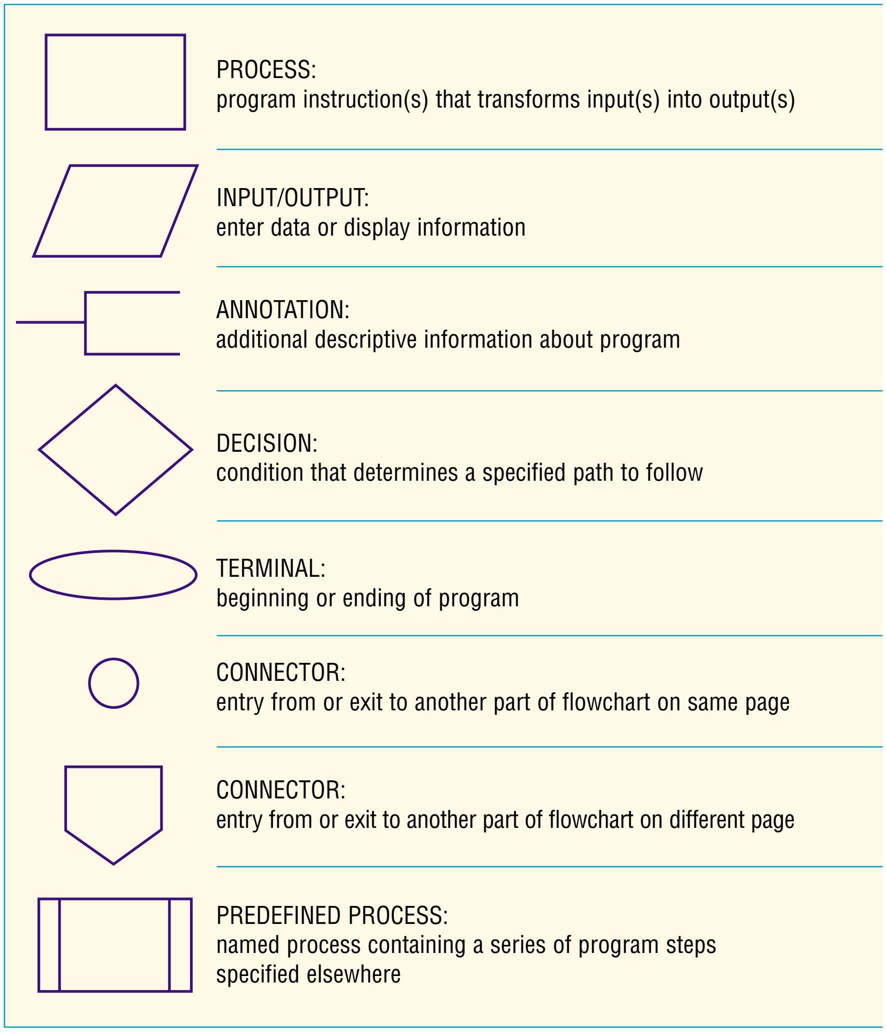

PROCESS:
program instruction(s) that transforms input(s) into output(s)
INPUT/OUTPUT:
enter data or display information
ANNOTATION:
additional descriptive information about program
DECISION:
condition that determines a specified path to follow
TERMINAL:
beginning or ending of program
CONNECTOR:
entry from or exit to another part of flowchart on same page
CONNECTOR:
entry from or exit to another part of flowchart on different page
PREDEFINED PROCESS:
named process containing a series of program steps specified elsewhere

Figure 15-11 Standard symbols used to create program flowcharts.

Web Link

For more information on flowcharting software, visit the Discovering Computers 2002 Chapter 15 WEB LINK page (**scsite.com/dc2002/ch15/weblink.htm**) and click Flowcharting Software.

Programmers connect most symbols on a program flowchart with solid lines. These lines show the direction of the program. Dotted lines on a flowchart connect comment symbols. A **comment symbol**, also called an **annotation symbol**, explains or clarifies logic in the solution algorithm. Figure 15-12 shows the program flowchart for three modules of the program shown in the hierarchy chart in Figure 15-4 on page 15.6.

In the past, programmers used a template to trace the symbols for a flowchart on paper. Today, programmers use commercial **flowcharting software** to develop flowcharts. This software makes it easy to modify and update flowcharts (Figure 15-13).

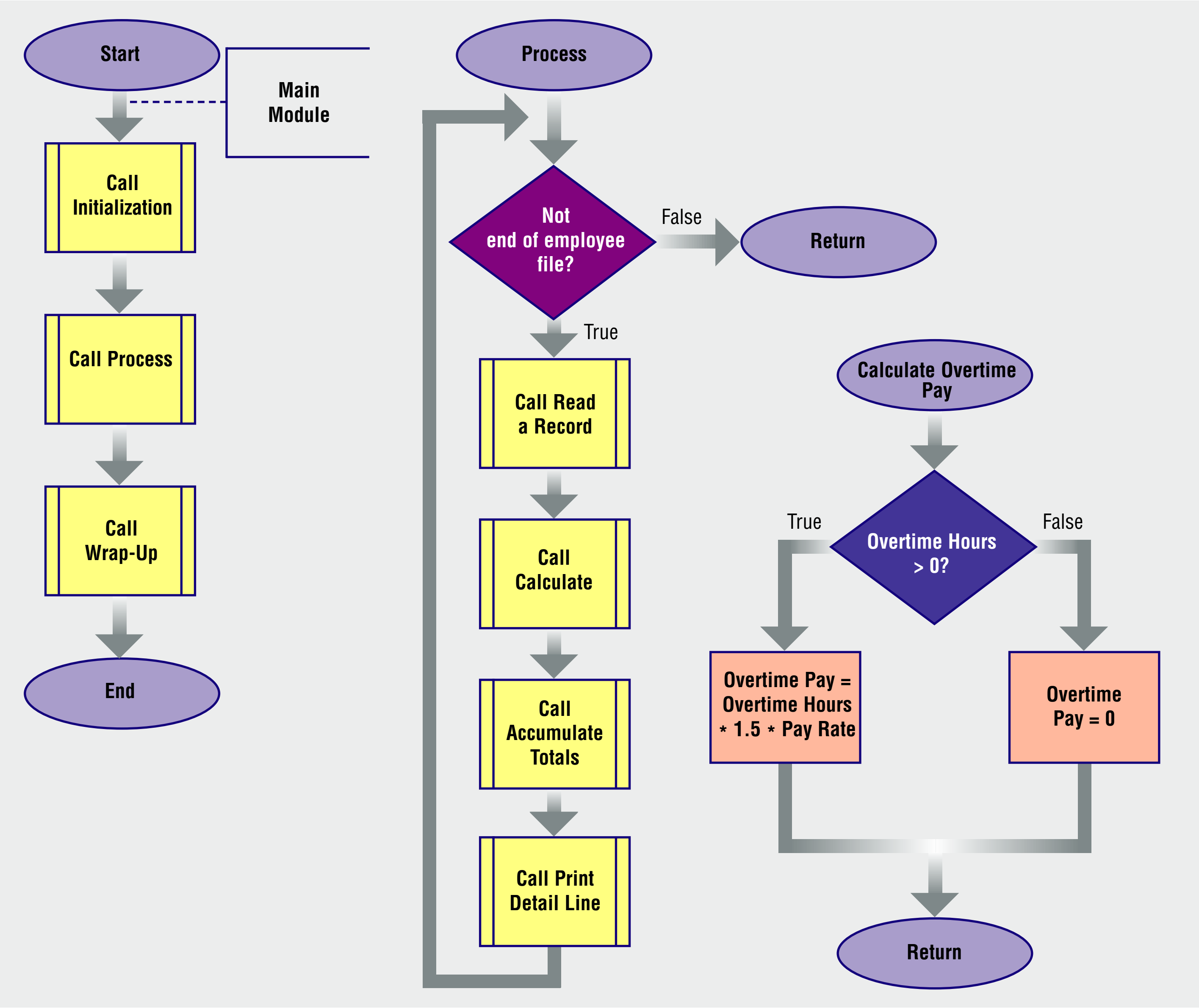


Figure 15-12 A program flowchart is drawn for each module on the hierarchy chart. Three modules from Figure 15-4 on page 15.6 are shown in this figure: main, process, and calculate overtime pay. Notice the main module is terminated with the word, End; whereas, the subordinate modules end with the word, Return, because they return to a higher-level module.

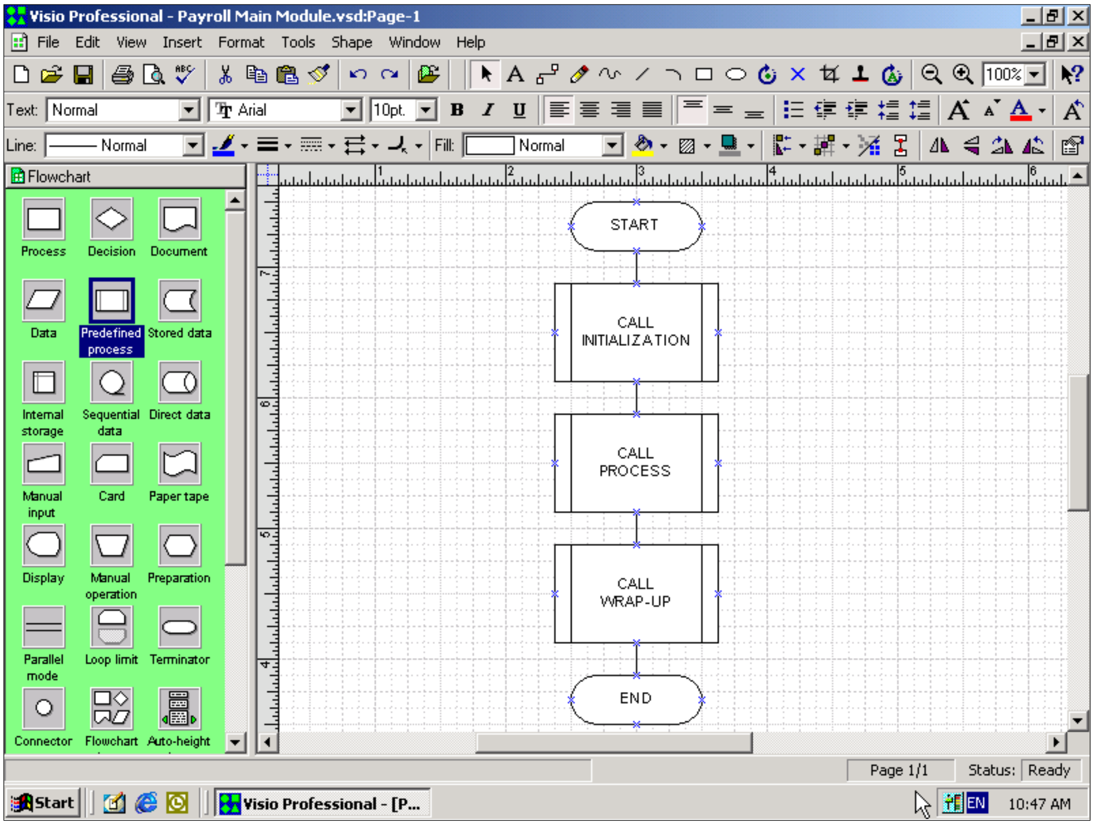


Figure 15-13 Visio is a popular flowcharting software package.

NASSI-SCHNEIDERMAN CHART A **Nassi-Schneiderman (N-S) chart** also graphically shows the logic in a solution algorithm. N-S charts sometimes are called **structured flowcharts** because they easily represent each of the three basic control structures. Unlike a program flowchart, an N-S chart does not use lines to show direction. Instead, N-S charts use a series of rectangular boxes, one below the next, with the flow always moving from top to bottom. Figure 15-14 shows N-S charts for the same three program modules shown as flowcharts in Figure 15-12.

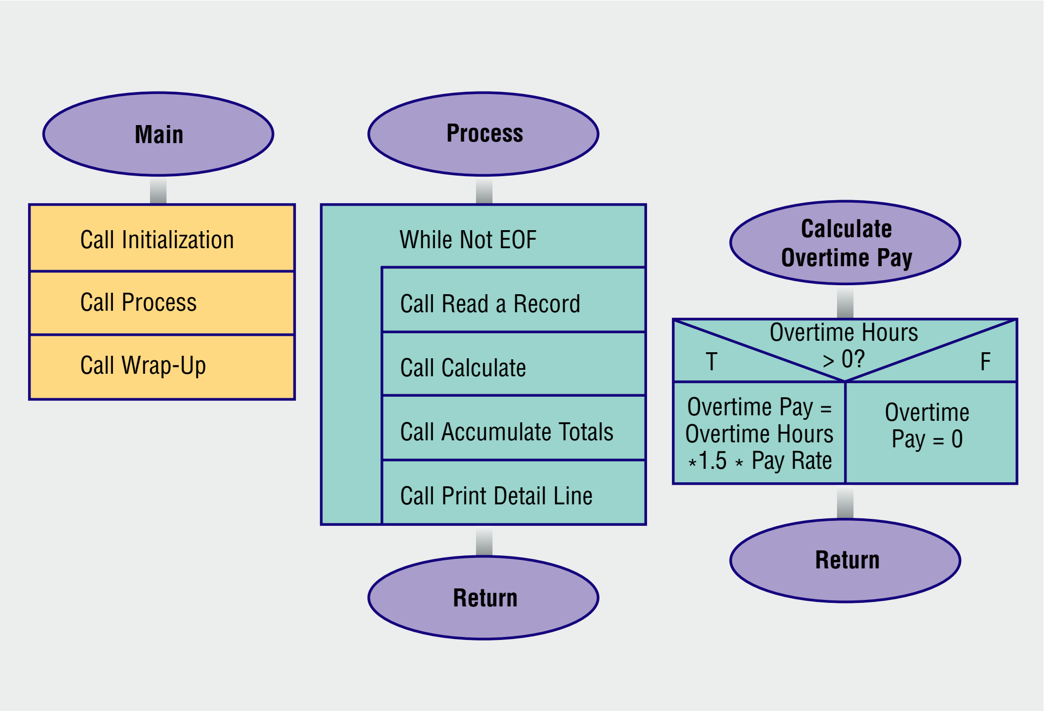


Figure 15-14 An N-S chart is an alternative method of showing program logic. This figure shows the same three modules (main, process, and calculate overtime pay) as illustrated in Figure 15-12 with program flowcharts.

PSEUDOCODE Some programmers prefer to explain the logic of a solution algorithm with words, instead of a graphical flowcharting technique. **Pseudocode** uses a condensed form of English to convey program logic. It also uses indentation to identify the three basic control structures. The beginning and ending of the module start at the left margin. You indent the actions within the module. The actions within a selection or repetition control structure are indented again. This allows you to identify clearly the beginning and ending of the control structure. Figure 15-15 shows the pseudocode for the same three program modules as in Figure 15-12 on page 15.10 and Figure 15-14 on the previous page.

```
MAIN MODULE:

     CALL Initialization
     CALL Process
     CALL Wrap-Up

END

PROCESS MODULE:

     DO WHILE Not EOF
        CALL Read a Record
        CALL Calculate
        CALL Accumulate Totals
        CALL Print Detail Line
     ENDDO

RETURN

CALCULATE OVERTIME PAY MODULE:

     IF Overtime Hours > 0 THEN
        Overtime Pay = Overtime Hours * 1.5 * Pay Rate
     ELSE
        Overtime Pay = 0
     ENDIF

RETURN
```

Figure 15-15 Pseudocode is yet another alternative method of showing program logic. This figure shows the same three modules (main, process, and calculate overtime pay) as illustrated in Figure 15-12 on page 15.10 with program flowcharts and Figure 15-14 on the previous page with N-S charts.

Quality Review Techniques

Once programmers develop the solution algorithm using a program flowchart, an N-S chart, or pseudocode, they should perform a quality review of the program. During a **quality review**, the programmer checks the logic for correctness and attempts to uncover logic errors. A **logic error** is a flaw in the design that causes inaccurate results. Two techniques for reviewing a solution algorithm are a desk check and a structured walkthrough.

When you **desk check** a program, you use test data to step through its logic. **Test data** is sample data that mimics data the program might process once it is in production. The programmer that developed the solution algorithm usually performs the desk check, but another programmer also can perform the desk check. Desk checking involves five steps.

1. Developing sets of test data (inputs)
2. Determining the expected result (output) for each set of data, without using the solution algorithm
3. Stepping through the solution algorithm using one set of test data and writing down the actual result obtained (output) using the solution algorithm
4. Comparing the expected result from Step 2 to the actual result from Step 3
5. Repeating Steps 3 and 4 for each set of test data

If the expected result and actual result do not match for any set of data, the program has a logic error. When this occurs, the programmer must review the logic of the solution algorithm to determine the reason for the error and then correct it.

A more formal technique for checking the solution algorithm is a structured walkthrough. As discussed in Chapter 14, a systems analyst often uses a structured walkthrough to review deliverables during the SDLC. Likewise, a programmer can request a walkthrough of a solution algorithm during the PDLC. In this case, the programmer explains the logic of the algorithm while members of the programming team step through the program logic. The purpose of this structured walkthrough is to identify errors in the program logic and check for possible improvements in program design.

Usually, a programmer easily can correct errors or improvements identified at this point. Once program design is complete and the programmer begins coding, errors are more difficult to fix. Thus, detecting errors and making improvements early in the PDLC reduces the overall time and cost of program development.

STEP 3 – CODE PROGRAMS

Coding a program involves two steps: (1) translating the solution algorithm into a programming language and (2) entering the programming language code into the computer. Of all tasks, programmers usually find coding the most tedious.

As previously mentioned, many different programming languages exist. Each of these has a particular syntax. A language's **syntax** is the set of grammar and rules that specifies how to write instructions for a solution algorithm. For example, a programmer writes an instruction to add three numbers differently in each language, according to its syntax.

As you enter a program into the computer, you should take time to document the program code thoroughly. Thus far, the programmer has created external program documentation in the form of flowcharts, N-S charts, or pseudocode. A program also has its own documentation, called **comments** or **remarks**.

Programs should include both global and internal comments (Figure 15-16). **Global comments** explain the program's purpose and identify the program name, its author, and the date written. Generally, global comments are at the top of the program. Internal comments, by contrast, appear throughout the body of the program. **Internal comments** explain the purpose of the code statements within the program. Programs that include global and internal comments are much easier to maintain.

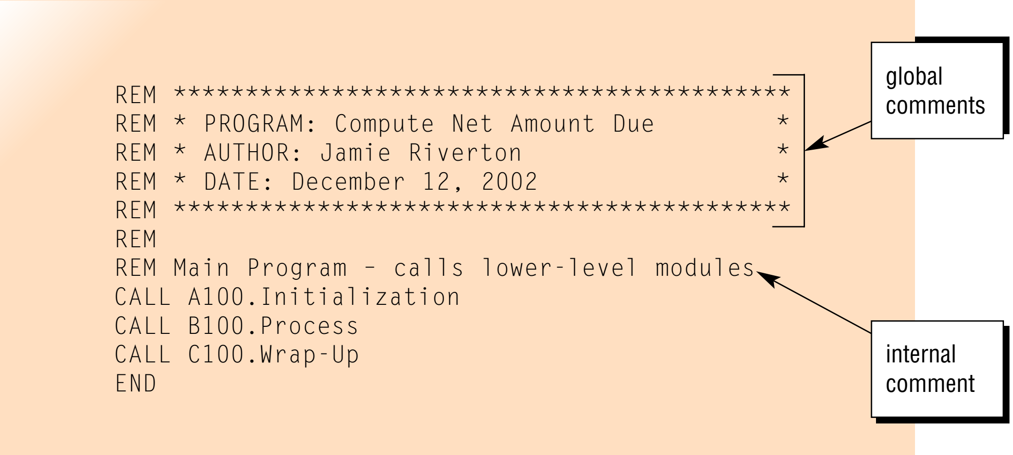
STEP 4 – TEST PROGRAMS

Once a programmer codes and enters the program, the next step is to test it. Thorough testing is very important. Once the program is put into use, many users rely on the program and its output to support their daily activities and decisions. Users should assist in the development of test data.

The goal of program testing is to ensure the program runs correctly and is error free. Errors uncovered during this step usually are one of two types: (1) syntax errors or (2) logic errors.

A **syntax error** occurs when the code violates the syntax, or grammar, of the programming language. Misspelling a command, leaving out required punctuation, or typing command words out of order all will cause syntax errors. You usually discover syntax errors the first time you execute the program code on the computer. When a syntax error is located, a message either displays on the screen immediately or is written to a log file. Either way, the programmer must review and correct all syntax errors. The procedure for testing for logic errors at this step is much like the desk checking techniques used in the Design Programs step. First, you develop test data. In the Design Programs step, the programmer develops the test data. In this Test Programs step, by contrast, the systems analyst usually develops the test data. The test data should include both valid (correct) and invalid (incorrect) input data. When valid test data is input, the program should produce the correct result. If the expected result and actual result do not match, the program has a logic error. In this case, the programmer must review the logic of the program code to determine the cause of the logic error and then correct it.

Another purpose of using test data is to try to crash the system. For example, the programmer will deliberately try to cause **run time errors**, or make the program fail while it is running. If the pay rate for employees cannot exceed $55 per hour, then the test data should use some valid pay rates, such as $25 and $10.50, as well as some invalid ones, such as $-32.00 and $72.50. When you input an invalid pay rate, the program should display an error message and allow you to re-enter the pay rate. If, however, the program accepts an invalid pay rate, it contains a logic error.



```
REM ******************************************
REM * PROGRAM: Compute Net Amount Due        *
REM * AUTHOR: Jamie Riverton                 *
REM * DATE: December 12, 2002                *
REM ******************************************
REM
REM Main Program - calls lower-level modules
CALL A100.Initialization
CALL B100.Process
CALL C100.Wrap-Up
END
```

global comments
internal comment

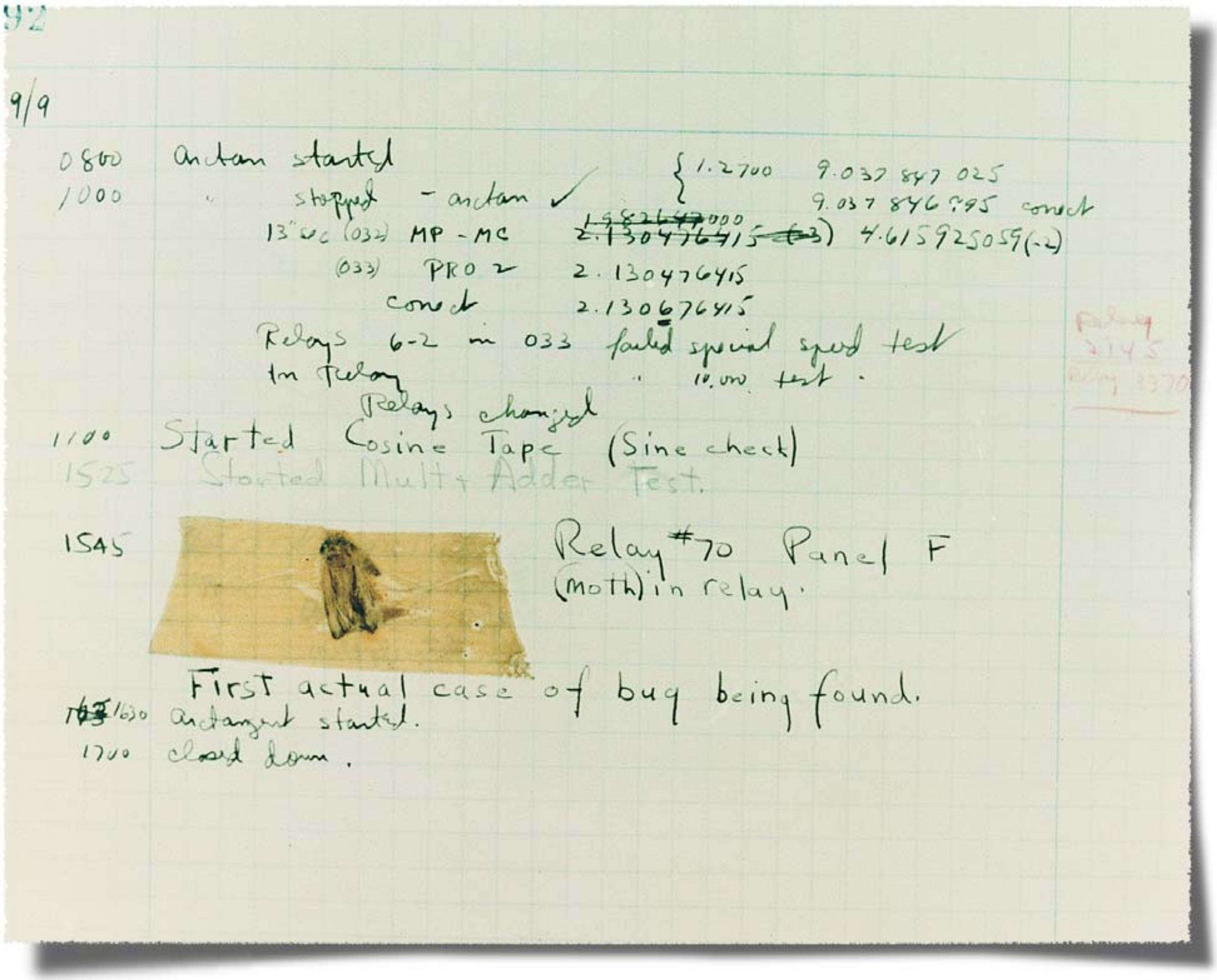
Figure 15-16 Thorough documentation leads to maintainable programs. A program should contain global comments (at the top of the program) and internal comments (throughout the body of the program). In this QuickBASIC program, comments are identified by the letters, REM, which are an abbreviation for REMARK.

The process of locating and correcting syntax and logic errors in a program is known as **debugging** the program. The errors themselves are the bugs. Thus, removing the errors is de*bug*ging. It is said that the term bug originated when the failure of one of the first computers was traced to an actual bug. A moth lodged in the computer's electronic components was the cause of the failure (Figure 15-17).

Most programming languages include a debug utility. A **debug utility**, or **debugger**, allows you to identify syntax errors and to find logic errors. With these utilities, you can examine program values (such as the result of a calculation) while the program runs in slow motion.

A bug that had the potential to cause serious financial losses for computers around the world was the millennium bug. The **millennium bug**, or **Y2K bug**, took effect when the computer date rolled over to January 1, 2000. At this time, non-Y2K compliant computers read the date as 01/01/00. Computers that used just the last two digits of a year could not distinguish 01/01/1900 from 01/01/2000. Most companies prepared themselves thoroughly for the Y2K bug and made necessary corrections to dates in hardware and software well in advance of January 1, 2000. Even with proper preparation, some computer hardware and software had problems.

If a program is well designed during the Design Programs step, then testing should not require much time. When a programmer does not test the solution algorithm thoroughly during design, many logic errors may exist and testing can take longer. As a general rule, the more time and effort programmers spend analyzing and designing the solution algorithm, the less time they will spend debugging the program.



92

9/9

0800 Arctan started
1000 " stopped - arctan ✓ {1.2700 9.037 847 025
9.037 846 995 conect
13" uc (032) MP-MC ~~1.982147000~~ ~~2.130476415~~ ~~(-3)~~) 4.615925059(-2)
(033) PRO 2 2.130476415
conect 2.130676415
Relays 6-2 in 033 failed special speed test
In relay " 10,000 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult+ Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
~~143~~1630 Antangent started.
1700 closed down.

Relay
2145
Relay 3370

Figure 15-17 The temporary failure of one of the first computers was traced to a dead moth (shown taped to the log book) caught in the electrical components. Some say this event is the origin of the term bug, which means computer error.

STEP 5 – FORMALIZE SOLUTION

In formalizing the solution, the programmer performs two activities: (1) review the program code and (2) review all the documentation.

First, programmers review the program for any dead code and remove it. Then, they should run the program one final time to verify it still works.

After reviewing the program code, the programmer gives the program and all of its documentation to the systems analyst. The documentation includes the following: a hierarchy chart; a solution algorithm in the form of a program flowchart, an N-S chart, or pseudocode; test data; and program code listings that contain global and internal comments.

The programmer should be sure all documentation is complete and accurate. This becomes especially valuable if the program requires changes at a later date. One year later, a new programmer might have to update these programs. Proper documentation greatly reduces the amount of time a new programmer spends learning about existing programs.

STEP 6 – MAINTAIN PROGRAMS

Maintaining programs involves two activities: (1) correcting errors and (2) adding enhancements. Once programs are **implemented**, or placed into production, users interact with the programs. These programs process actual, also known as live, transactions.

During the course of their use, programs may require maintenance. One type of maintenance occurs when users encounter program errors. If the programmer and systems analyst thoroughly tested the solution algorithm and program code in the earlier steps, the number of errors found during production should be very small.

A more common type of maintenance occurs when a user would like the program to have new features or functionality. **Program enhancement** involves modifying existing programs to improve their functionality.

When users identify errors or enhancements, they typically notify the systems analyst. The systems analyst then contacts and meets with a programmer. Sometimes, the systems analyst meets with the programmer who wrote the original program. If the programmer is not available, the systems analyst meets with a different programmer. During the initial meeting, the systems analyst and the programmer begin analyzing the problem or enhancement, which is Step 1 of the PDLC. The PDLC thus completes its loop and begins again.

ISSUE

Coping with Bugs

Huge Costs of Computer Bugs

The consequences of bugs in computer programs can be staggering. One mistake in the code controlling a Canadian nuclear facility caused more than 3,000 gallons of radioactive water to be spilled. A bug in AT&T's long-distance switching software cost the company $60 million. Despite sophisticated debugging utilities, experts estimate that one in every 5,000 lines of code contains an error. Given that many programs contain hundreds of thousands, even millions, of code lines, is it possible to remove all the bugs from a program? Why or why not? What can be done to reduce the number of software bugs? If software bugs are inevitable, or at least to be expected depending on the complexity of the code, what steps can people relying on computer programs take to deal with the bugs? Should programmers be held responsible for these bugs?

For more information on debugging programs, visit the Discovering Computers 2002 Issues Web page (**scsite.com/dc2002/issues.htm**) and click Chapter 15 Issue #2.

PROGRAMMING LANGUAGES AND PROGRAM DEVELOPMENT TOOLS

A computer programmer can select from a variety of programming languages or program development tools to create solutions to information system requirements. A **programming language** is a set of words, symbols, and codes that enables a programmer to communicate a solution algorithm to the computer. Just as humans understand a variety of spoken languages (English, Spanish, French, and so on), you can use a variety of programming languages to code a program.

A **program development tool** consists of user-friendly software products designed to assist both programmers and nontechnical users with the creation of information system solutions. If you use a program development tool, you often do not need to learn a programming language. Program development tools automatically generate the programming language instructions necessary to communicate with the computer.

The following sections discuss programming languages and program development tools.

CATEGORIES OF PROGRAMMING LANGUAGES

Several hundred programming languages exist today. Each language has its own syntax, or rules. Some languages work with only certain computers. Other languages are for specific purposes, such as scientific or business applications. Many languages, however, run on several types of computers. These programming languages often follow standards set by the American National Standards Institute (ANSI). ANSI standard programs can run on many different types of computers, as well as many different operating systems.

Five major categories of programming languages exist: machine languages, assembly languages, third-generation languages, fourth-generation languages, and fifth-generation languages. These languages are classified as either low level or high level.

A **low-level language** is a programming language that is machine dependent. A **machine-dependent language** only runs on one particular computer. These programs are not portable to other computers. Machine and assembly languages are low-level languages.

A **high-level language**, by contrast, is a language that is machine independent. A **machine-independent language** can run on many different types of computers and operating systems. Third-generation, fourth-generation, and fifth-generation languages are high-level languages.

The following sections discuss each of the five major categories of programming languages.

Machine Languages (First-Generation Language)

Machine language is the first generation of programming languages. The only language the computer directly understands is **machine language** (Figure 15-18). Machine language instructions use a series of binary digits (1s and 0s) or a combination of numbers and letters that represent binary digits. The binary digits correspond to the on and off electrical states of a computer. As you might imagine, coding in machine language can be tedious and time-consuming.

Assembly Languages (Second-Generation Languages)

Assembly languages are the second generation of programming languages. With an **assembly language**, a programmer writes instructions using symbolic instruction codes (Figure 15-19). **Symbolic instruction codes**, or **mnemonics**, are meaningful abbreviations and codes. With an assembly language, a programmer writes codes such as A for addition, C for compare, L for load, and M for multiply.

Assembly languages also use symbolic addresses. A **symbolic address** is a meaningful name that identifies a storage location. For example, a programmer can use the name RATE to refer to the storage location that contains the pay rate.

Despite these advantages, assembly languages can be difficult to learn. In addition, you must convert an assembly language program into machine language before the computer can understand it. The computer cannot understand or execute the assembler source program. A **source program** is the program that contains the assembly language code. To convert the assembly language source program into machine language, you use an **assembler**.

One assembly language instruction usually translates into one machine language instruction. In some cases, however, the assembly language includes macros. This type of **macro** generates multiple machine language instructions for a single assembly language instruction. Macros save the programmer time during program development.

```
                                        00090
000090  50E0    30B2                    010B4
000094  1B44
000096  1B77
000098  1B55
00009A  F273    30D6    2C81    010D8   00C83
0000A0  4F50    30D6                    010D8
0000A4  F275    30D6    2C7B    010D8   00C7D
0000AA  4F70    30D6                    010D8
0000AE  5070    304A                    0104C
0000B2  1C47
0000B4  5050    304E                    01050
0000B8  58E0    30B2                    010B4
0000BC  07FE
                                        000BE
0000BE  50E0    30B6                    010B8
0000C2  95F1    2C85            00C87
0000C6  4770    20D2            000D4
0000CA  1B55
0000CC  5A50    35A6                    015A8
0000D0  47F0    2100            00102
0000D4  95F2    2C85            00C87
0000D8  4770    20E4            000E6
0000DC  1B55
0000DE  5A50    35AA                    015AC
0000E2  47F0    2100            00102
000102  1B77
000104  5870    304E                    01050
000108  1C47
00010A  4E50    30D6                    010D8
00010E  F075    30D6    003E    010D8   0003E
000114  4F50    30D6                    010D8
000118  5050    3052                    01054
00011C  58E0    30B6                    010B8
000120  07FE
                                        00122
000122  50E0    30BA                    010BC
000126  1B55
000128  5A50    304E                    01050
00012C  5B50    3052                    01054
000130  5050    305A                    0105C
000134  58E0    30BA                    010BC
000138  07FE
```

Figure 15-18 A sample machine language program.

```
*         THIS MODULE CALCULATES THE REGULAR TIME PAY
CALCSTPY  EQU    *
          ST     14,SAVERTPY
          SR     4,4
          SR     7,7
          SR     5,5
          PACK   DOUBLE,RTHRSIN
          CVB    4,DOUBLE
          PACK   DOUBLE,RATEIN
          CVB    7,DOUBLE
          ST     7,RATE
          MR     4,7
          ST     5,RTPAY
          L      14,SAVERTPY
          BR     14
*         THIS MODULE CALCULATES THE OVERTIME PAY
CALCOTPY  EQU    *
          ST     14,SAVEOTPY
TEST1     CLI    CODEIN,C'O'
          BH     TEST2
          SR     5,5
          A      5,=F'0'
          ST     5,OTPAY
          B      AROUND
TEST2     SR     4,4
          SR     7,7
          SR     5,5
          PACK   DOUBLE,OTHRSIN
          CVB    4,DOUBLE
          PACK   DOUBLE,RATEIN
          CVB    7,RATE
          MR     4,7
          MR     4,=F'1.5'
          ST     5,OTPAY
AROUND    L      14,SAVEOTPY
          BR     14
*         THIS MODULE CALCULATES THE GROSS PAY
CALCGPAY  EQU    *
          ST     14,SAVEGPAY
          SR     5,5
          A      5,RTPAY
          A      5,OTPAY
          ST     5,GRPAY
          L      14,SAVEGPAY
          BR     14
```

Figure 15-19 An excerpt from an assembly language program. The code shows the computations for regular time pay, overtime pay, and gross pay, and the decision to evaluate the overtime hours.

Third-Generation Languages

The disadvantages of low-level machine and assembly languages led to the development of third generation languages in the late 1950s and 1960s. A programmer uses a series of English-like words to write a **third-generation language** (**3GL**) instruction. For example, ADD stands for addition or PRINT means to print. Many 3GLs also use arithmetic operators such as * for multiplication and + for addition. These English-like words and arithmetic symbols simplify the program development process for the programmer.

Third-generation languages are procedural languages. A **procedural language** requires the program instructions tell the computer *what* to accomplish and *how* to do it. With most 3GLs a programmer can use both the top-down approach (modules) and structured constructs (sequence, selection, and repetition) within each module to develop these procedures.

Like an assembly language program, the 3GL code is called the source program. You must convert this source program into machine language before the computer can understand it. This translation process often is very complex, because one 3GL source program instruction translates into many machine language instructions. For 3GLs, you typically use either a compiler or an interpreter to perform this translation.

A **compiler** converts the entire source program into machine language before executing it. The machine language version that results from compiling the 3GL is called the **object code** or **object program**. The compiler stores the object code on disk for execution at a later time.

While it is compiling the source program into object code, the compiler checks the source program's syntax. It also makes sure the program properly defines the data it will use in calculations or comparisons. The compiler then produces a program listing, which contains the source code and a list of any syntax errors. This listing helps the programmer make necessary changes to the source code and debug the program. Figure 15-20 shows the process of compiling a source program.

A compiler translates an entire program before executing it. An interpreter, by contrast, translates and executes one program code statement at a time. An **interpreter** reads a code statement, converts it to one or more machine language instructions, and then executes those machine language instructions. It does this all before moving to the next code statement in the program. Each time you run the source program, the interpreter translates and executes it, statement by statement. An interpreter does not produce an object program. Figure 15-21 shows the process of interpreting a program.

One advantage of an interpreter is it immediately displays feedback when it finds a syntax error. The programmer can correct the error, or debug the code, before the interpreter translates the next line. The disadvantage is that interpreted programs do not run as fast as compiled programs. This is because an interpreter must translate the source program to machine language each time you execute it. Once you compile a program, however, you simply execute the object code to run the program.

Many programming languages include both an interpreter and a compiler. In this case, the programmer can use the interpreter to debug the program. Once debugged, the programmer can compile the program so it runs faster when it is placed into production.

```
*    COMPUTE REGULAR TIME PAY
     MULTIPLY REGULAR-TIME-HOURS BY HOURLY-PAY-RATE
         GIVING REGULAR-TIME-PAY.

*    COMPUTE OVERTIME PAY
     IF OVERTIME-HOURS > 0
          COMPUTE OVERTIME-PAY = OVERTIME-HOURS * 1.5 * HOURLY-PAY-RATE
     ELSE
         MOVE 0 TO OVERTIME-PAY.

*    COMPUTE GROSS PAY
     ADD REGULAR-TIME-PAY TO OVERTIME-PAY
         GIVING GROSS-PAY.

*    PRINT GROSS PAY
     MOVE GROSS-PAY TO GROSS-PAY-OUT.
     WRITE REPORT-LINE-OUT FROM DETAIL-LINE
         AFTER ADVANCING 2 LINES.
```

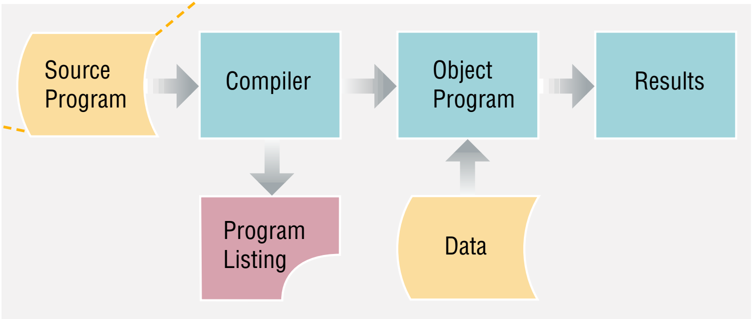


Figure 15-20 With a compiler, the entire source program is converted into a machine language object program. If the compiler encounters any errors, it records them in the program-listing file, which the programmer may print when the entire compilation is complete. When the user wants to run the program, the object program is loaded into the memory of the computer and the program instructions begin executing.

Fourth-Generation Languages

Like a 3GL, a **fourth-generation language** (**4GL**) uses English-like statements. The syntax of 4GL, however, is closer to human language than that of a 3GL. A 4GL is a nonprocedural language. With a **nonprocedural language**, the programmer only specifies *what* the program should accomplish without explaining how. Consequently, programmers spend much less time and effort coding programs with a 4GL. These languages tend to be quite easy to use. Thus, users with very little programming background also can develop programs using a 4GL.

Many 4GLs work with a database and its project dictionary. These powerful languages allow database administrators to define the database and its structure. They also help users maintain and query the data in the database. As discussed in Chapter 13, SQL is a popular ANSI-standard 4GL used with relational DBMSs (Figure 15-22). Recall that SQL is a **query language** enabling users and programmers to retrieve data from database tables. One query, for example, might request a list of all employees receiving overtime pay.

Some DBMSs provide a report writer. As discussed in Chapter 13, a **report writer** or **report generator** is software that allows you to design a report on the screen, retrieve data into the report design, and then display or print the report. Behind the scenes, these report writers build a 4GL query that enables you to access the data.

One advantage of a report writer is that you can retrieve data without having to learn a query language. Another advantage is you easily can include page numbers and dates; report titles and column headings; subtotals and totals; numeric formatting (such as dollars and cents); and other formatting features such as fonts, font sizes, color, and shading. Report writers usually are menu-driven and have a graphical user interface.

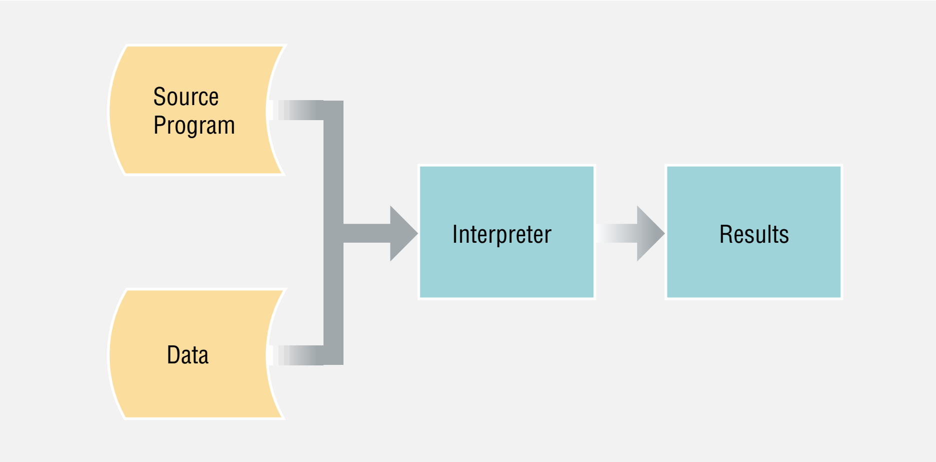


Figure 15-21 With an interpreter, one line of the source program at a time is converted into machine language and then immediately executed by the computer. If the interpreter encounters an error while converting a line of code, an error message immediately displays on the screen and the interpretation stops.

```
SELECT LAST_NAME, FIRST_NAME, GROSS_PAY

FROM EMPLOYEE

WHERE OVERTIME_HOURS > 0

ORDER BY LAST_NAME;

LAST_NAME     FIRST_NAME     GROSS_PAY

Antiqua       Martin            780.00
Charles       Leslie            715.00
Guillan       Anita             847.50
.
.
.
```

Figure 15-22 SQL is a fourth-generation language that can be used to query database tables. This query produces an alphabetical list of those employees who receive overtime pay; that is, their overtime hours are greater than zero.

Fifth-Generation Languages

A **fifth-generation language** (**5GL**) is one that provides a visual or graphical interface for creating the source code. The 5GL often converts the source code to machine language using a 3GL or a 4GL compiler. Object-oriented and Web development tools sometimes use a 5GL. Visual Basic, which is illustrated in Figure 15-25 on page 15.23, is a 5GL.

ISSUE

What's the Secret?

Sharing Program Code

Linux is a fast-growing, innovative new operating system. Linux differs from other operating systems because of its open source code, which means there are no proprietary restrictions to the operating system code. Anyone may alter the code to his or her own needs without asking for, or receiving, any permission. Something else that makes Linux different from other operating systems is that the program is downloadable right from the Web, and at no cost. Since its introduction in 1991, hundreds of programmers, each adding their expertise to make the code better, have altered, adapted, and improved Linux. While Linux is an open-source program, application code for other operating systems is a zealously guarded secret. At a large software developer firm, an employee claims that system programmers have little opportunity to contribute to application programs because they have no access to the application program source code. Why are many commercial software developers reluctant to share program code? What are the advantages of making program code public? What are the disadvantages?

For more information on open source code, visit the Discovering Computers 2002 Issues Web page (**scsite.com/dc2002/issues.htm**) and click Chapter 15 Issue #3.

OBJECT-ORIENTED PROGRAM DEVELOPMENT

As discussed earlier in this chapter, the introduction of structured program design and its three basic constructs largely solved the problem of spaghetti code. Structured program design, however, does not provide a way to keep the data and the program (or procedure) together. Each program has to define how it will use the data for a particular program. This can result in redundant programming code that must change every time the structure of the data changes. To eliminate this problem, some IT professionals use the object-oriented approach for program development. With the **object-oriented** (**OO**) **approach**, the programmer can package the data and the program (or procedure) into a single unit, called an object. When the structure of an object changes, any program that accesses the object automatically accesses the change.

An **object** is an item that can contain both data and the procedures that read or manipulate that data. An Employee object might contain data about an employee (Employee ID, First Name, Last Name, Address, and so on) and instructions on how to print the employee record or the formula required to calculate the employee's gross pay. The procedure in the object, called a **method** or **operation**, contains activities that read or manipulate the data. The data elements are called **attributes** or **variables**. For example, automobile manufacturers can program their assembly line cars (objects) to send messages to paint booths asking for an available slot and color (attributes).

Encapsulation, also called **information hiding**, is the concept of packaging methods and attributes into a single object. That is, an object encapsulates, or *hides*, the details of the object from the programmer. Think of an object as a box and you cannot see inside the box. The box sends and receives messages. It also contains code and data. Users do not need to look into the box because the object package is self-sufficient. When you want to print a document, you click the Print button (the object). You probably do not know the Print button actually communicates with the hardware to print your document. Thus, the details of your print object are encapsulated, or hidden from you. Programmers, however, need to know how the object works so they can send messages to it and use it effectively.

An object may be part of a larger category of objects, called a **class**. Every object in a class shares methods and attributes that are part of the original object. Each class can have one or more lower levels called **subclasses** (Figure 15-23). For example, ink-jet printer, laser printer, and thermal printer are all subclasses of a higher-level class, called printer. The higher-level class is called a **superclass**. Each subclass inherits the methods and attributes of the objects in its superclass. All printers have a weight (which would be an attribute in the printer object), but only ink-jet printers have ink-filled print cartridges. This concept of lower levels inheriting methods and attributes of higher levels is called **inheritance**.

A specific occurrence of an object or object class is called an **object instance**. For example, a Hewlett-Packard DeskJet printer in your home is an object instance of the ink-jet printer object.

To make an object do something, you send it a message. A **message** tells the object what to do. It indicates the name of the method to be used. For example, print a document might be a method of the printer object.

A major benefit of the OO approach is the ability to reuse and modify existing objects. If a program had to test the performance of 20 printers, a programmer could reuse the printer object over and over. The OO approach saves programming time when developing programs.

With the OO approach, the development team uses different analysis, design, and programming techniques and tools from those used in structured program development. Many developers today use Unified Modeling Language. **UML** (**Unified Modeling Language**) contains the standard notation for analysis, design, and documentation for the OO approach. Several CASE tools now include UML notations. Recall from Chapter 14 that a systems analyst uses CASE tools to automate activities in the system development process.

In the late 1990s, the Object Management Group adopted UML as a standard. The **Object Management Group** (**OMG**) is an international organization that establishes guidelines and specifications for OO application development. In addition to UML, the OMG also has adopted CORBA as a programming standard. **CORBA** (**Common Object Request Broker Architecture**) defines how objects in separate programs on a network can communicate with each other.

Object-Oriented Programming

With the OO approach, a programmer uses an **object-oriented programming** (**OOP**) **language** to implement the OO design. An OOP language is event-driven. **Event** is simply the OOP term for message. An event-driven program checks for and responds to a set of events. An event could be pressing a key on the keyboard, clicking the mouse, or typing a value into a text box. Some programming languages are not OOP languages, but they are event-driven. Others, such as C++, are complete object-oriented languages. The following section covers specific examples of these and other programming languages in more detail.

Web Link

For more information on object-oriented programming languages, visit the Discovering Computers 2002 Chapter 15 WEB LINK page (**scsite.com/dc2002/ch15/weblink.htm**) and click Object-Oriented Programming Languages.

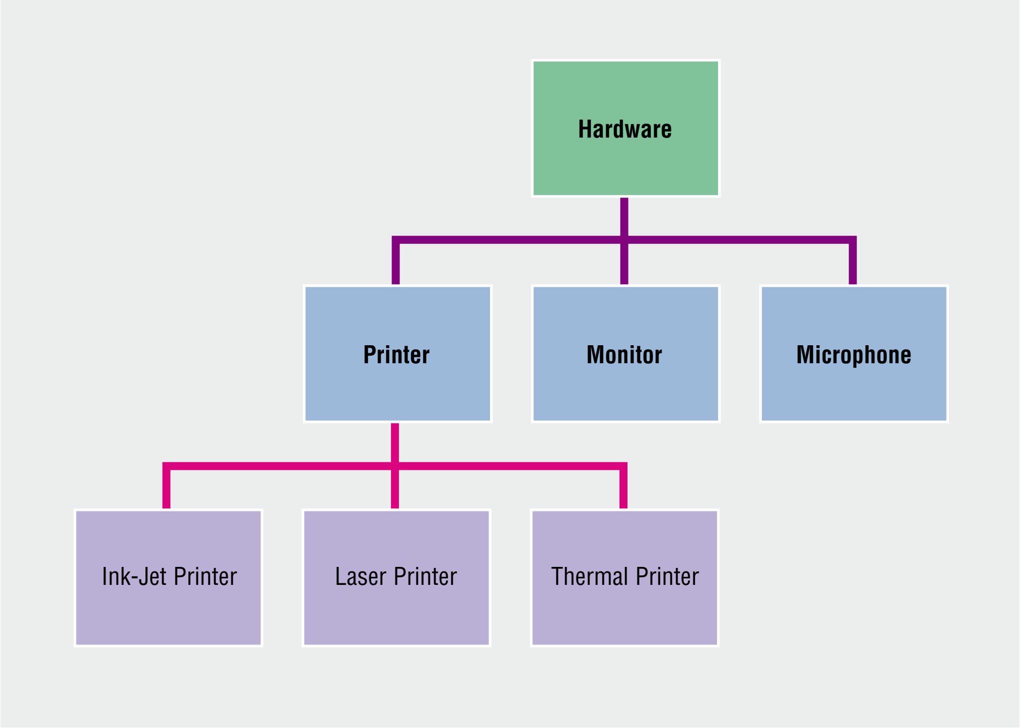


Figure 15-23 A generalization hierarchy is an object-oriented analysis tool that shows relationships between classes of an object. In this figure, printer has three subclasses. A subclass inherits (or acquires) the attributes and methods of its higher class.

PROGRAMMING LANGUAGES

Hundreds of programming languages exist. Only a few, however, are used widely enough today for the industry to recognize them as standards. Most of these are high-level languages that work on a variety of computers. This section discusses these programming languages, their origins, and their primary purpose. Languages used for Web page development and multimedia applications are discussed later in the chapter.

To illustrate the similarities and differences among programming languages, figures on the following pages show program code in several programming languages. The code solves a simple payroll problem — computing the gross pay for an employee. The steps to compute gross pay can vary from one system to another. The examples on the following pages use a simple algorithm to help you easily compare one programming language to another.

To compute the gross pay, first you multiply the regular time hours worked by the hourly rate of pay to obtain the regular time pay. If the employee has overtime hours, their overtime pay is 1.5 times their hourly rate of pay multiplied by overtime hours. Then, you add the regular time pay and overtime pay together.

BASIC

John Kemeny and Thomas Kurtz developed a programming language called **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode, or **BASIC**, in the mid-1960s at Dartmouth College. Kemeny and Kurtz designed BASIC for use as a simple, interactive problem-solving language (Figure 15-24). BASIC originally was intended as, and sometimes is still, the language used in a student's first programming course because it is so easy to learn and use. Today, BASIC is used on both personal computers and minicomputers to develop some business applications. Many versions of BASIC exist, including QBasic, QuickBASIC, and MS-BASIC.

Visual Basic

Developed by Microsoft Corporation in the early 1990s, **Visual Basic** is a Windows-based application that assists programmers in developing other event-driven Windows-based applications. The first step in building a Visual Basic application is to design the graphical user interface using Visual Basic objects (Steps 1 and 2 in Figure 15-25). Visual Basic objects, or controls, include items such as command buttons, text boxes, and labels.

Next, you write any code needed to define program events (Step 3 in Figure 15-25). An event in Visual Basic might be the result of an action initiated by a user. When a user

```
REM COMPUTE REGULAR TIME PAY
Regular.Time.Pay = Regular.Time.Hours * Hourly.Pay.Rate

REM COMPUTE OVERTIME PAY
If Overtime.Hours > 0 THEN
    Overtime.Pay = Overtime.Hours * 1.5 * Hourly.Pay.Rate
ELSE
    Overtime.Pay = 0
END IF

REM COMPUTE GROSS PAY
Gross.Pay = Regular.Time.Pay + Overtime.Pay

REM PRINT GROSS PAY
PRINT USING "The gross pay is $##,###.##"; Gross.Pay
```

Figure 15-24 This figure shows an excerpt from a BASIC program. The code shows the computations for regular time pay, overtime pay, and gross pay; the decision to evaluate the overtime hours; and the output of the gross pay.

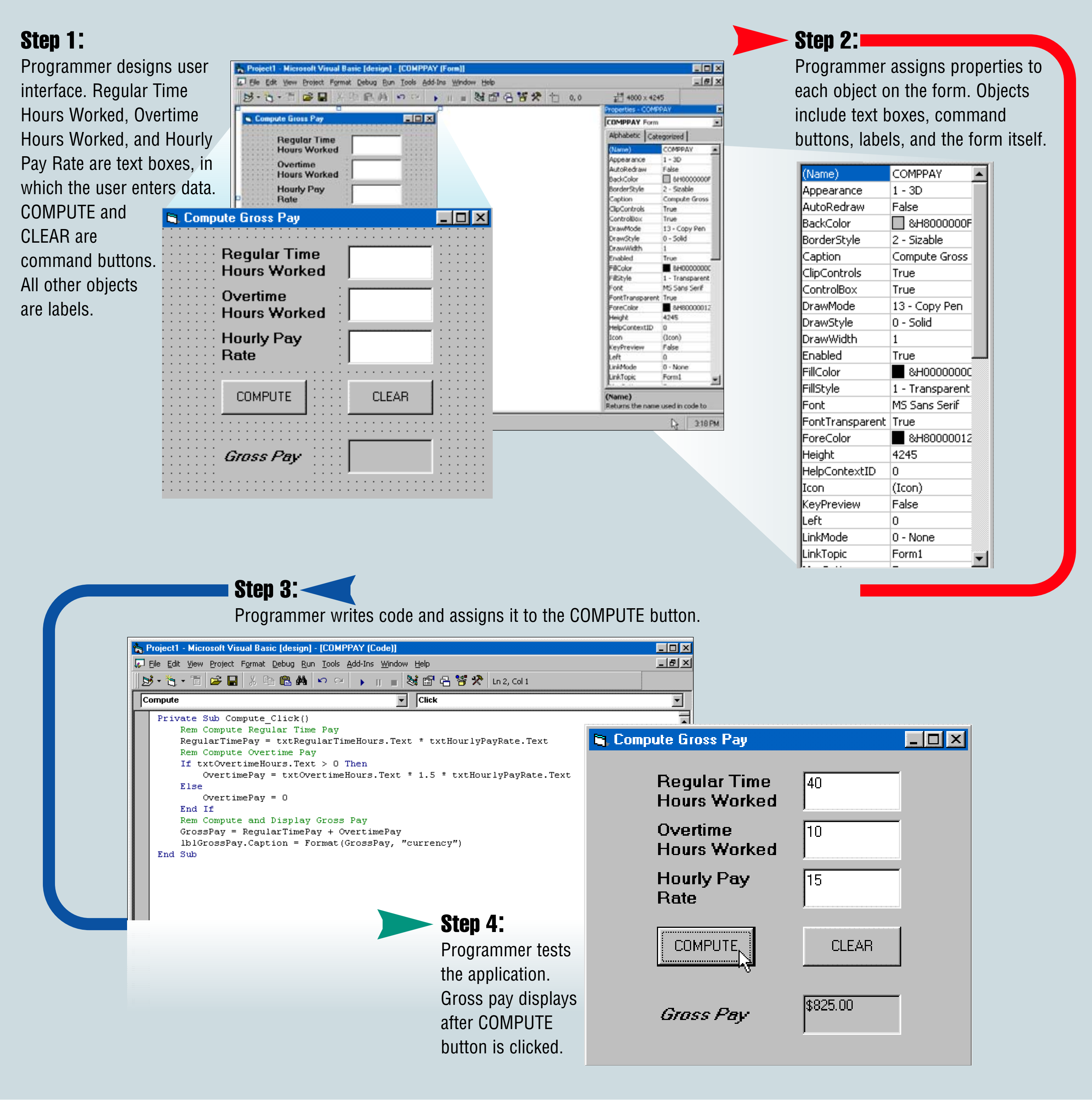
clicks an object in a Visual Basic application, the application executes the Click event. You define Visual Basic events using code statements written in Visual Basic's own programming language. This language is very similar to BASIC and easy to learn and use. Once you have completed these steps, you can generate and test the final application (Step 4 in Figure 15-25).

Beginning programmers can create professional Windows-based applications using Visual Basic because the program is easy to use.

Web Link

For more information on Visual Basic, visit the Discovering Computers 2002 Chapter 15 WEB LINK page (**scsite.com/dc2002/ch15/weblink.htm**) and click Visual Basic.

Figure 15-25 CREATING A VISUAL BASIC APPLICATION



Step 1:
Programmer designs user interface. Regular Time Hours Worked, Overtime Hours Worked, and Hourly Pay Rate are text boxes, in which the user enters data. COMPUTE and CLEAR are command buttons. All other objects are labels.

Project1 - Microsoft Visual Basic [design] - [COMPPAY (Form)]
Compute Gross Pay
Regular Time Hours Worked
Overtime Hours Worked
Hourly Pay Rate
COMPUTE
CLEAR
Gross Pay

Step 2:
Programmer assigns properties to each object on the form. Objects include text boxes, command buttons, labels, and the form itself.

(Name)	COMPPAY
Appearance	1 - 3D
AutoRedraw	False
BackColor	&H8000000F
BorderStyle	2 - Sizable
Caption	Compute Gross
ClipControls	True
ControlBox	True
DrawMode	13 - Copy Pen
DrawStyle	0 - Solid
DrawWidth	1
Enabled	True
FillColor	&H0000000C
FillStyle	1 - Transparent
Font	MS Sans Serif
FontTransparent	True
ForeColor	&H80000012
Height	4245
HelpContextID	0
Icon	(Icon)
KeyPreview	False
Left	0
LinkMode	0 - None
LinkTopic	Form1

Step 3:
Programmer writes code and assigns it to the COMPUTE button.

Project1 - Microsoft Visual Basic [design] - [COMPPAY (Code)]
Compute
Click
Ln 2, Col 1

```
Private Sub Compute_Click()
    Rem Compute Regular Time Pay
    RegularTimePay = txtRegularTimeHours.Text * txtHourlyPayRate.Text
    Rem Compute Overtime Pay
    If txtOvertimeHours.Text > 0 Then
        OvertimePay = txtOvertimeHours.Text * 1.5 * txtHourlyPayRate.Text
    Else
        OvertimePay = 0
    End If
    Rem Compute and Display Gross Pay
    GrossPay = RegularTimePay + OvertimePay
    lblGrossPay.Caption = Format(GrossPay, "currency")
End Sub
```

Step 4:
Programmer tests the application. Gross pay displays after COMPUTE button is clicked.

Compute Gross Pay

Regular Time Hours Worked: 40
Overtime Hours Worked: 10
Hourly Pay Rate: 15
Gross Pay: $825.00

COBOL

COBOL (**CO**mmon **B**usiness-**O**riented **L**anguage) developed out of a joint effort between the United States government, businesses, and major universities in the early 1960s. Naval officer Grace Hopper, a pioneer in computer programming, was a prime developer of the COBOL language.

COBOL is one of the more widely used procedural programming languages for business applications. Although COBOL programs often are lengthy, their English-like statements make the code easy to read, write, and maintain (Figure 15-26). COBOL is especially useful for processing transactions on mainframes. COBOL programs also run on other types of computers. The most popular personal computer COBOL program is MERANT Micro Focus Net Express, which allows you to create procedural and object-oriented COBOL programs and migrate them to the Web.

```
*    COMPUTE REGULAR TIME PAY
     MULTIPLY REGULAR-TIME-HOURS BY HOURLY-PAY-RATE
         GIVING REGULAR-TIME-PAY.

*    COMPUTE OVERTIME PAY
     IF OVERTIME-HOURS > 0
         COMPUTE OVERTIME-PAY = OVERTIME-HOURS * 1.5 * HOURLY-PAY-RATE
     ELSE
         MOVE 0 TO OVERTIME-PAY.

*    COMPUTE GROSS PAY
     ADD REGULAR-TIME-PAY TO OVERTIME-PAY
         GIVING GROSS-PAY.

*    PRINT GROSS PAY
     MOVE GROSS-PAY TO GROSS-PAY-OUT.
     WRITE REPORT-LINE-OUT FROM DETAIL-LINE
         AFTER ADVANCING 2 LINES.
```

Figure 15-26 An excerpt from a COBOL program. The code shows the computations for regular time pay, overtime pay, and gross pay; the decision to evaluate the overtime hours; and the output of the gross pay. Notice how much wordier this program is compared with the BASIC program in Figure 15-24 on page 15.22. It is, however, much more readable than the BASIC program.

TECHNOLOGY TRAILBLAZER

GRACE **HOPPER**

Imagine getting this assignment on your first day at a new job: "have the coefficients for the interpolation of the arc tangents by next Thursday." That was Grace Hopper's first mission upon graduating from Yale University in 1934. And she had to perform these calculations on the Mark I "computer engine," which measured 51 feet by 8 feet by 8 feet and performed three additions per second.

In the next years she found herself programming the Mark I to find the angles to aim naval guns in varying weather conditions, building and programming the Navy's Mark II and III computers, and building the UNIVAC I computer. She is credited with finding the first computer "bug" shown in Figure 15-17 on page 15.14.

In an effort to make programming easier for laypeople, she developed the first compiler. She also created the FLOW-MATIC programming language, which used English phrases in business applications. This language laid the foundation for COBOL.

For more information on Grace Hopper, visit the Discovering Computers 2002 People Web page (**scsite.com/dc2002/people.htm**) and click Grace Hopper.

C

The **C** programming language, developed in the early 1970s by Dennis Ritchie at Bell Laboratories, originally was designed for writing system software. Today, a variety of software programs are written in C. This includes operating systems and application software such as word processing and spreadsheet programs.

C is a powerful language that requires professional programming skills. Many programmers use C for business and scientific problems (Figure 15-27). C runs on almost any type of computer with any operating system, but it most often is used with the UNIX operating system. In fact, most of the UNIX operating system is written in C.

C++

Developed in the 1980s by Bjarne Sroustrup at Bell Laboratories, **C++** (pronounced SEE-plus-plus) is an object-oriented programming language. C++ is an extension of the C programming language. It includes all the elements of the C language plus has additional features for working with objects, classes, events, and other object-oriented concepts. Programmers commonly use C++ to develop application software, such as word processing and spreadsheet programs, as well as database and Web applications. Although C++ is an outgrowth of the C programming language, you do not need C programming experience to be a successful C++ programmer.

Some programmers use a newer programming language called C# (pronounced SEE-sharp). **C#** combines features of C and C++ and is best suited for development of Web applications.

```
/* Compute Regular Time Pay                                  */
rt_pay = rt_hrs * pay_rate;

/* Compute Overtime Pay                                      */
if (ot_hrs > 0)
    ot_pay = ot_hrs * 1.5 * pay_rate;
else
    ot_pay = 0;

/* Compute Gross Pay                                         */
gross = rt_pay + ot_pay

/* Print Gross Pay                                           */
printf("The gross pay is %d\n", gross);
```

Figure 15-27 An excerpt from a C or C++ program. The code shows the computations for regular time pay, overtime pay, and gross pay; the decision to evaluate the overtime hours; and the output of the gross pay.

TECHNOLOGY TRAILBLAZER

DENNIS **RITCHIE**

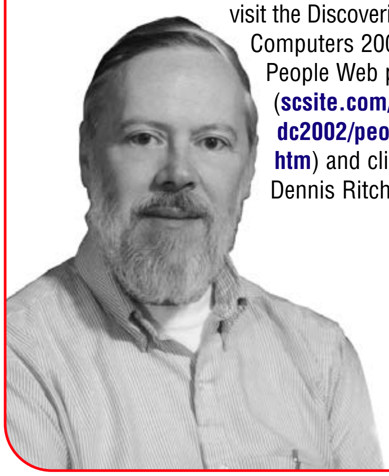
Children learn their ABCs; programmers learn their BCC++s. Many of today's Internet applications and commercial software packages are written using C++, a useful object-oriented language. C++ is an expansion of C, which Dennis Ritchie developed at Bell Labs in the early 1970s.

At that time, systems programmers were using the UNIX operating system on meager Digital Equipment Corporation PDP-11 computers with 24 KB of memory and a programming language called B, which was developed by Ken Thompson. Ritchie tweaked B to make it portable, so it could be installed easily on computers other than the PDP-11 without extensive software rewrites. C became extremely popular outside of Bell Labs after 1980.

President Bill Clinton awarded Ritchie and Thompson the National Medal of Technology, the country's highest prize for achievement in technology, in 1999.

Besides being a powerful force behind Bell Labs' Computer Sciences Research Center, Ritchie also drove a tractor-trailer for the Roush Racing Valvoline/Cummings NASCAR team and served as a mechanic on Mark Martin's racecar and a front tire carrier during pit stops.

For more information on Dennis Ritchie, visit the Discovering Computers 2002 People Web page (**scsite.com/dc2002/people.htm**) and click Dennis Ritchie.



Web Link

For more information on C++, visit the Discovering Computers 2002 Chapter 15 WEB LINK page (**scsite.com/dc2002/ch15/weblink.htm**) and click C++.

RPG

In the early 1960s, IBM introduced **RPG**, which stands for **R**eport **P**rogram **G**enerator, to assist businesses in generating reports (Figure 15-28). Today, businesses also use RPG for complex computations and file updating. Because RPG is a nonprocedural language, many users of RPG (and other report generators) claim it paved the way for 4GLs. RPG primarily is used for application development on IBM midrange computers, such as the AS/400. A version with limited functionality is available for the personal computer.

```
C* COMPUTE REGULAR TIME PAY
C          RTHRS     MULT RATE            RTPAY     72
C*
C* COMPUTE OVERTIME PAY
C          OTHRS     IFGT 0
C          RATE      MULT 1.5             OTRATE    72
C          OTRATE    MULT OTHRS           OTPAY     72
                     ELSE
C                    INZ                  OTPAY     72
C
C* COMPUTE GROSS PAY
C          RTPAY     ADD  OTPAY           GRPAY     72
C
C* PRINT GROSS PAY
C                    EXCPTDETAIL
C
C*
O* OUTPUT SPECIFICATIONS
OQPRINT  E           DETAIL
O                                  23 'THE GROSS PAY IS $'
O                    GRPAY  J       34
```

Figure 15-28 This figure shows an excerpt from an RPG program. The code shows the computations for regular time pay, overtime pay, and gross pay; the decision to evaluate the overtime hours; and the output of the gross pay.

Other Programming Languages

In addition to the programming languages just discussed, programmers sometimes do use other languages. Figure 15-29 lists some of these languages and their primary applications.

ADA	Derived from Pascal, developed by the U.S. Department of Defense, named after Augusta Ada Lovelace Byron, who is thought to be the first female computer programmer
ALGOL	ALGOrithmic Language, the first structured procedural language
APL	A Programming Language, a scientific language designed to manipulate tables of numbers
FORTH	Similar to C, used for device control applications
FORTRAN	FORmula TRANSlator, one of the first high-level programming languages used for scientific applications
HYPERTALK	An object-oriented programming language developed by Apple to manipulate cards that can contain text, graphics, and sound
LISP	LISt Processing, a language used for artificial intelligence applications
LOGO	An educational tool used to teach programming and problem-solving to children
MODULA-2	A successor to Pascal used for developing systems software
PASCAL	Developed to teach students structured programming concepts, named in honor of Blaise Pascal, who developed one of the earliest calculating machines
PILOT	Programmed Inquiry Learning Or Teaching, used to write computer-aided instruction programs
PL/I	Programming Language One, a business and scientific language that combines many features of Fortran and COBOL
PROLOG	PROgramming LOGic, used for development of artificial intelligence applications
SMALLTALK	Object-oriented programming language

Figure 15-29 Other programming languages.

ISSUE

Is Generic Okay?

One Programming Language for all Applications

Different languages are used for different applications because each programming language has its strengths and weaknesses. Some authorities believe, however, it would be best to develop a single programming language that could be used for all applications. SQL is a language that is probably one of the most standardized, but even within SQL there are variations. What would be the advantages of writing all applications using the same programming language? What would be the disadvantages? Do you think such a generic programming language ever will be developed? Why or why not?

For more information on programming languages, visit the Discovering Computers 2002 Issues Web page (**scsite.com/dc2002/issues.htm**) and click Chapter 15 Issue #4.

PROGRAM DEVELOPMENT TOOLS

Program development tools are user-friendly software products designed to help both program developers and nontechnical users create solutions to information requirements. With these tools, IT professionals develop systems faster. Program development tools also **empower** nontechnical users by giving them the ability to write simple programs and satisfy information requests on their own. This allows programmers and other IT professionals to focus development efforts on larger projects.

Examples of program development tools include query languages and report writers, which were discussed earlier in this chapter and Chapter 13. In addition to query languages and report writers, other software development tools include application generators, macros, and RAD tools.

Application Generators

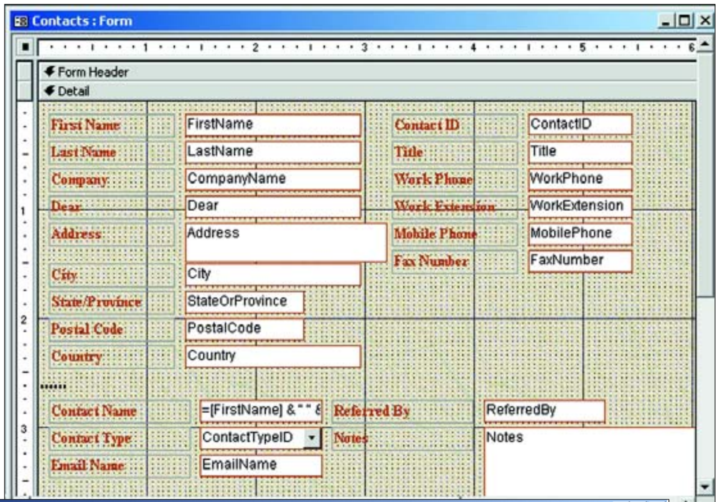
An **application generator**, sometimes called a **program generator**, is a program that allows you to build an application without writing the extensive code required by a 3GL. Programmers that use application generators can be more productive in a shorter time frame. In addition, many users that are unfamiliar with programming concepts can build applications with an application generator. Application generators empower users by providing them with the ability to create applications on their own.

When using an application generator, the developer (a programmer or user) works with menu-driven tools that have graphical user interfaces. Some application generators create source code. Others simply create object code. Some application generators are available as stand-alone programs. Most often they are bundled with or part of a DBMS. An application generator typically includes a report writer, form, and menu generator.

As discussed in Chapter 13, a report writer allows you to design a report on the screen, retrieve data into the report design, and then display or print the report. A form is a window on the screen that provides areas for entering or changing data in a database.

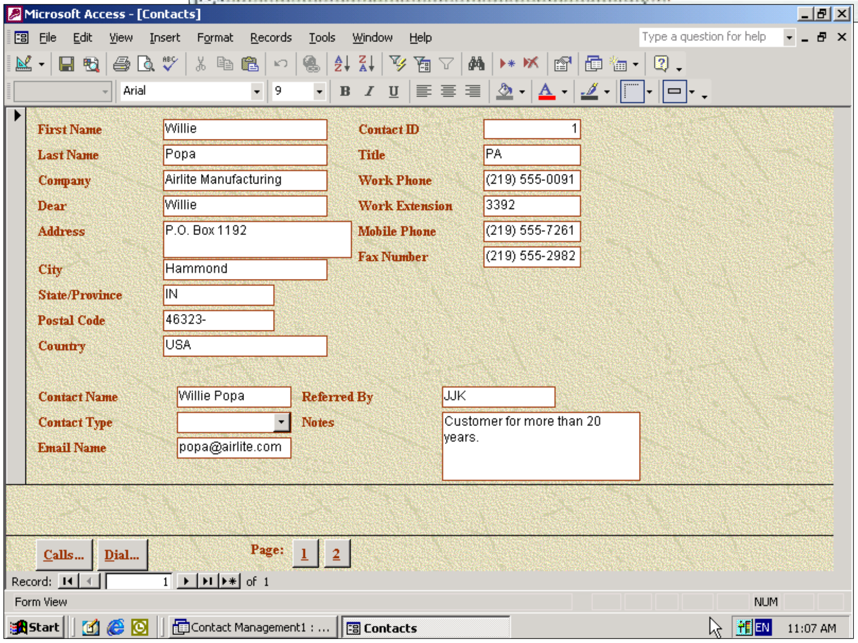
With both a report writer and a form, developers type titles and headings directly on the screen. Then, they describe the data elements to be used usually by clicking the location where the data elements should display or print. The data dictionary contains characteristics about the data elements. When the developer uses an existing data element, the report writer or form accesses the data dictionary for the format specifications of the data element. Figure 15-30 shows a sample form design and the resulting form it generates.

Figure 15-30a (form design)



First Name	FirstName	Contact ID	ContactID
Last Name	LastName	Title	Title
Company	CompanyName	Work Phone	WorkPhone
Dear	Dear	Work Extension	WorkExtension
Address	Address	Mobile Phone	MobilePhone
City	City	Fax Number	FaxNumber
State/Province	StateOrProvince		
Postal Code	PostalCode		
Country	Country		
Contact Name	=[FirstName] & " " &	Referred By	ReferredBy
Contact Type	ContactTypeID	Notes	Notes
Email Name	EmailName		

Figure 15-30b (resulting form)



Microsoft Access - [Contacts]
File Edit View Insert Format Records Tools Window Help
Type a question for help
Arial
9
Calls...
Dial...
Page: 1 2
Record: 1 of 1
Form View
NUM
Start
Contact Management1 : ...
Contacts
EN
11:07 AM

First Name	Willie	Contact ID	1
Last Name	Popa	Title	PA
Company	Airlite Manufacturing	Work Phone	(219) 555-0091
Dear	Willie	Work Extension	3392
Address	P.O. Box 1192	Mobile Phone	(219) 555-7261
City	Hammond	Fax Number	(219) 555-2982
State/Province	IN		
Postal Code	46323-		
Country	USA		
Contact Name	Willie Popa	Referred By	JJK
Contact Type		Notes	Customer for more than 20 years.
Email Name	popa@airlite.com		

Figure 15-30 A form design and the resulting form created with Microsoft Access.

A **menu generator** allows the developer to create a menu, or list of choices, for the application options. If you create three reports and two forms for an application, for example, your menu would contain at least six options: one for each report, one for each form, and one to exit, or quit, the application.

Macros

Empowered users also can create simple programs within applications by writing macros. A **macro** is a series of statements that instructs an application how to complete a task. Macros can automate routine, repetitive, or difficult tasks in an application such as word processing, spreadsheet, or database programs. You usually create a macro in one of two ways: (1) record the macro or (2) write the macro.

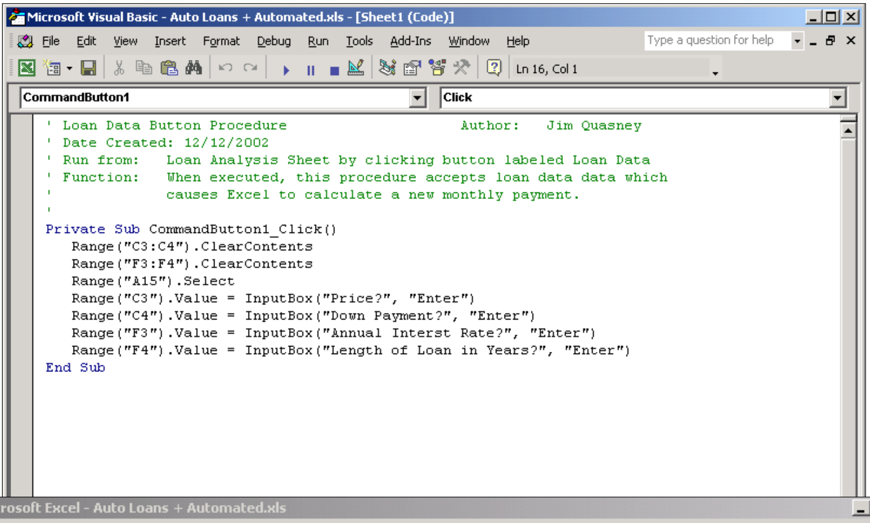
If you want to automate a routine or repetitive task such as formatting or editing, you would record a macro. A **macro recorder** is similar to a movie camera because both record all actions until turned off.

To record a macro, start the macro recorder in the application. Then, perform the steps to be part of the macro, such as clicks of the mouse or keystrokes. Once the macro is recorded, you can run it any time you want to perform that same set of actions. For example, you could record the actions required to format a number as a subscript. To change a selected number in a document to a subscript, you would run the Format Number As Subscript macro.

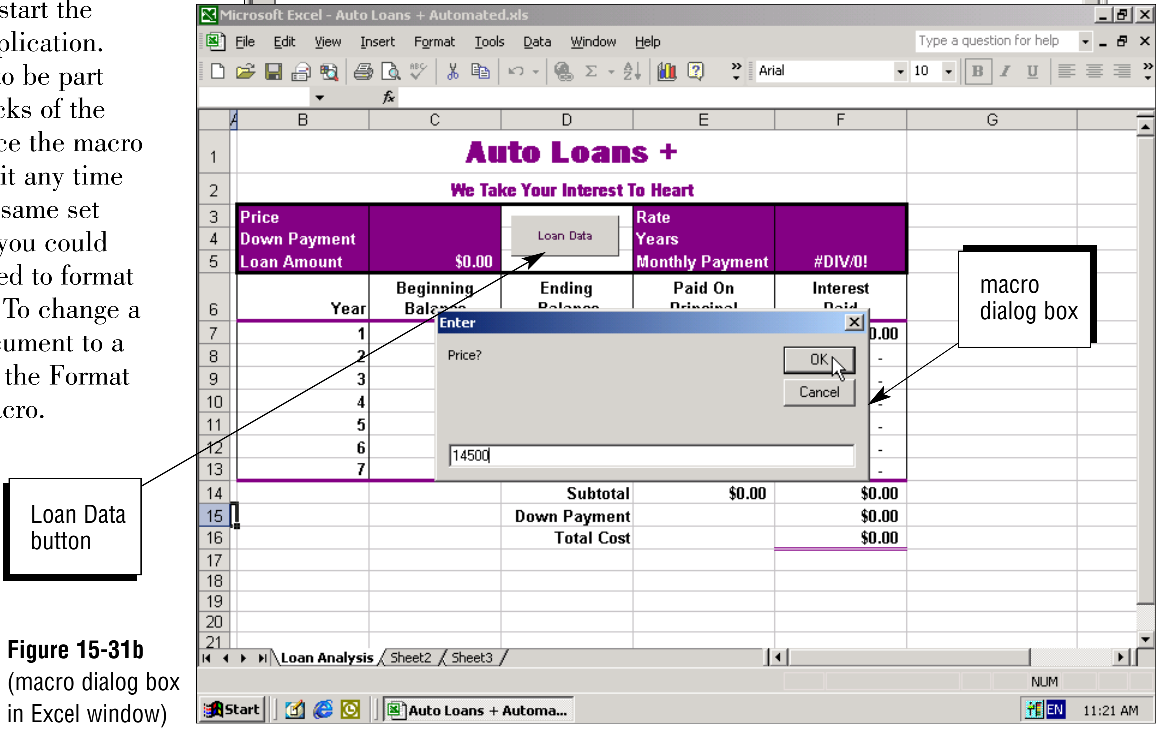
Once familiar with programming techniques, you can write your own macros instead of recording them. Many applications use **Visual Basic for Applications** (**VBA**) or a similar language as their macro programming language. Macros written in VBA use the three basic structured programming constructs (sequence, selection, and iteration) within modules. They also use objects, classes, and other object-oriented concepts. The objects in a VBA macro apply only to the specific application for which the macro was developed. In a spreadsheet, an object might be a cell, a range of cells, a chart, or the workbook.

The macro in Figure 15-31a shows an Excel macro that automates the data entry process to determine the interest and principal for a car loan. Figure 15-31b shows the dialog box generated from the macro that prompts the user to enter the purchase price of the vehicle.

Figure 15-31a (VBA macro)


Microsoft Visual Basic - Auto Loans + Automated.xls - [Sheet1 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Type a question for help
Ln 16, Col 1
CommandButton1
Click

```
' Loan Data Button Procedure                       Author:   Jim Quasney
' Date Created: 12/12/2002
' Run from:   Loan Analysis Sheet by clicking button labeled Loan Data
' Function:   When executed, this procedure accepts loan data data which
'             causes Excel to calculate a new monthly payment.
'
Private Sub CommandButton1_Click()
   Range("C3:C4").ClearContents
   Range("F3:F4").ClearContents
   Range("A15").Select
   Range("C3").Value = InputBox("Price?", "Enter")
   Range("C4").Value = InputBox("Down Payment?", "Enter")
   Range("F3").Value = InputBox("Annual Interst Rate?", "Enter")
   Range("F4").Value = InputBox("Length of Loan in Years?", "Enter")
End Sub
```



Auto Loans +

We Take Your Interest To Heart

Price
Down Payment
Loan Amount $0.00
Rate
Years
Monthly Payment #DIV/0!

Year | Beginning Balance | Ending Balance | Paid On Principal | Interest Paid
1
2
3
4
5
6
7

Subtotal $0.00 $0.00
Down Payment $0.00
Total Cost $0.00

Enter
Price?
14500

Figure 15-31b (macro dialog box in Excel window)

Figure 15-31 The top screen shows a VBA macro used to automate an auto loan computation. After a macro is written, the user clicks the Loan Data button to launch the macro. The bottom screen shows the macro guiding the user through the data entry process.

RAD Tools: Visual Basic, Delphi, and PowerBuilder

Rapid application development (**RAD**) is the concept of developing software throughout the system development process. Chapter 14 discussed that software development typically occurs in the implementation phase. RAD is an alternative to this technique.

A common technique of RAD is prototyping. As discussed in Chapter 14, a prototype is a working model of the proposed system. With a prototype, system developers can implement a solution more quickly than without a prototype. That is, they can develop the application rapidly.

Rapid application development uses RAD tools. These tools allow you to develop easy-to-maintain, component-based applications. **Component** is simply another term for object. A RAD tool typically includes an object-oriented programming language. Three popular RAD tools are Visual Basic, Delphi, and PowerBuilder.

VISUAL BASIC As discussed earlier in this chapter, Visual Basic is a Windows-based application that assists programmers in developing other event-driven Windows-based applications. Visual Basic was one of the first programming environments to provide a visual programming environment. A **visual programming environment** (**VPE**) allows developers to drag-and-drop objects to build programs. Visual Basic is a popular RAD tool because it is easy-to-use and event-driven.

DELPHI **Delphi** is another popular RAD tool. This powerful tool offers a drag-and-drop VPE. Delphi provides full object-oriented capabilities. This is different from Visual Basic, which is only an event-driven language. Delphi has more functionality and features than Visual Basic, making it slightly more difficult to learn and use.

Web Link

For more information on rapid application development, visit the Discovering Computers 2002 Chapter 15 WEB LINK page (**scsite.com/dc2002/ch15/weblink.htm**) and click Rapid Application Development.

APPLY IT!

Creating Macros in Microsoft Office

A macro consists of a series of commands that are grouped together as a single command. The single command is a convenient way to automate a difficult or lengthy task. Macros often are used for repetitive formatting or editing activities, to combine multiple commands into a single command, or to select an option within a dialog box with a single keystroke. To create macros in Microsoft Office 2000 requires the use of Visual Basic for Applications (VBA). This programming language is integrated into Access, Excel, FrontPage, Outlook, PowerPoint, and Word.

Within each of the above listed applications are two options for creating macros: record a macro or create a macro using the Visual Basic Editor. To record a macro, complete the following procedure:

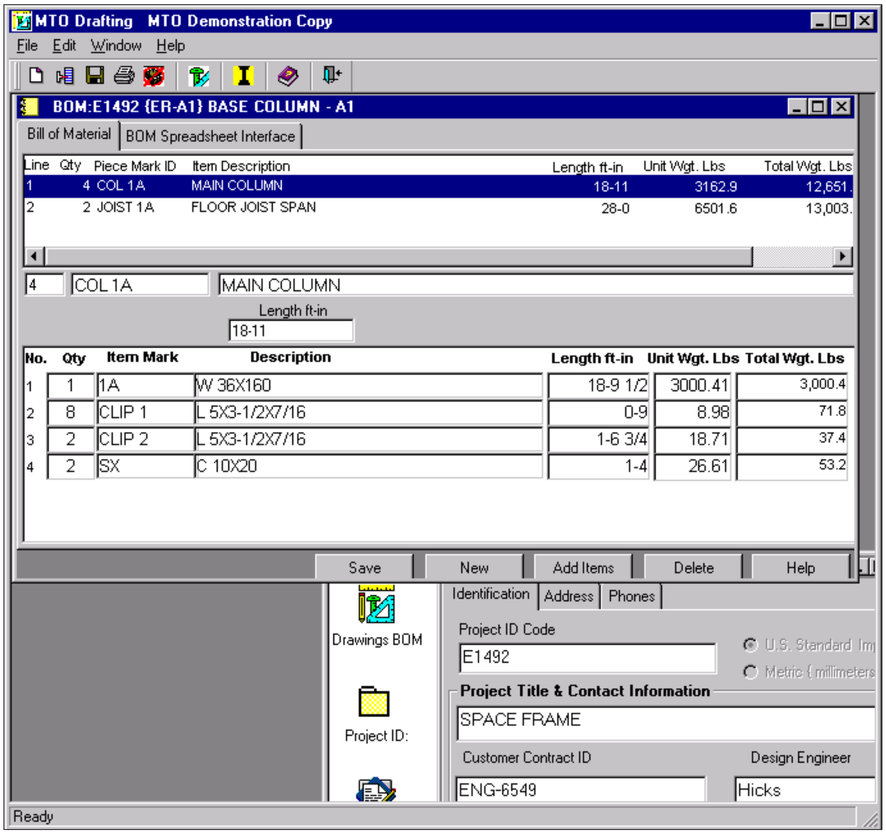
1. On the Tools menu, point to Macro, and then click Record New Macro.
2. Change the default macro name if you would like and then click the OK button to start the recorder.
3. Record the series of actions you want to automate.
4. Click the Stop Recording button on the Stop Recording toolbar.

To create a macro using the Visual Basic Editor requires that you know VBA programming commands. To create a macro with the Visual Basic Editor, complete the following procedure:

1. On the Tools menu, point to Macro, and then click Visual Basic Editor to open the VB Editor window.
2. Type your programming commands to create the series of actions you want to automate.
3. On the File menu, click Save Normal or Close and Return to [application name].

For more information on macros, visit the Discovering Computers 2002 Apply It Web page (**scsite.com/dc2002/apply.htm**) and click Chapter 15 Apply It #2.

POWERBUILDER Another RAD tool, called **PowerBuilder**, uses a proprietary object-oriented language for its application development (Figure 15-32). This language, called PowerScript, is similar to BASIC and C. PowerBuilder can create powerful applications. Many developers use PowerBuilder for Web-based applications. PowerBuilder can be more difficult to learn than Visual Basic because of its extended functionality.



Line	Qty	Piece Mark ID	Item Description	Length ft-in	Unit Wgt. Lbs	Total Wgt. Lbs
1	4	COL 1A	MAIN COLUMN	18-11	3162.9	12,651.
2	2	JOIST 1A	FLOOR JOIST SPAN	28-0	6501.6	13,003.

4 COL 1A MAIN COLUMN

Length ft-in: 18-11

No.	Qty	Item Mark	Description	Length ft-in	Unit Wgt. Lbs	Total Wgt. Lbs
1	1	1A	W 36X160	18-9 1/2	3000.41	3,000.4
2	8	CLIP 1	L 5X3-1/2X7/16	0-9	8.98	71.8
3	2	CLIP 2	L 5X3-1/2X7/16	1-6 3/4	18.71	37.4
4	2	SX	C 10X20	1-4	26.61	53.2

Project ID Code: E1492

● U.S. Standard Im
○ Metric (millimeters

Project Title & Contact Information: SPACE FRAME

Customer Contract ID: ENG-6549

Design Engineer: Hicks

Figure 15-32 Shown above is a material take off utility and bill of material application that is used in the structural steel industry. The application was created using PowerBuilder.

COMPANY ON THE CUTTING EDGE

SYBASE

PowerBuilder Users Get Top Technical Support

After waiting on hold for hours on the telephone listening to boring music, you are at your wits' end. You are having a problem with your software and just need to talk to a knowledgeable person for a few minutes.

Maybe if you would have chosen to work with the software programs PowerBuilder Enterprise, Powersoft, PowerStudio, or Adaptive Component Architecture, you might not be having this problem. What you need more than anything right now is a good customer support service. *InfoWorld* readers selected Sybase's online newsgroups as the *Best Customer Support* in the computer industry. These newsgroups are supported by Sybase support employees as well as dedicated customers.

No doubt some queries discuss the optimal method for using PowerBuilder, Sybase's RAD tool, because it helps developers efficiently create Internet applications and client/server programs. Developers at Blue Cross & Blue Shield in Rhode Island used this product to build a Y2K-compliant customer service system in record time. In addition, the National Marrow Donor Program in Minneapolis, MN, used Sybase products to create an application that searches a database and matches transplant donors with patients, and devoted tennis fans can watch the Sybase Open tournament via a Web site powered by Sybase technology.

For more information on Sybase, visit the Discovering Computers 2002 Companies Web page (**scsite.com/dc2002/companies.htm**) and click Sybase.

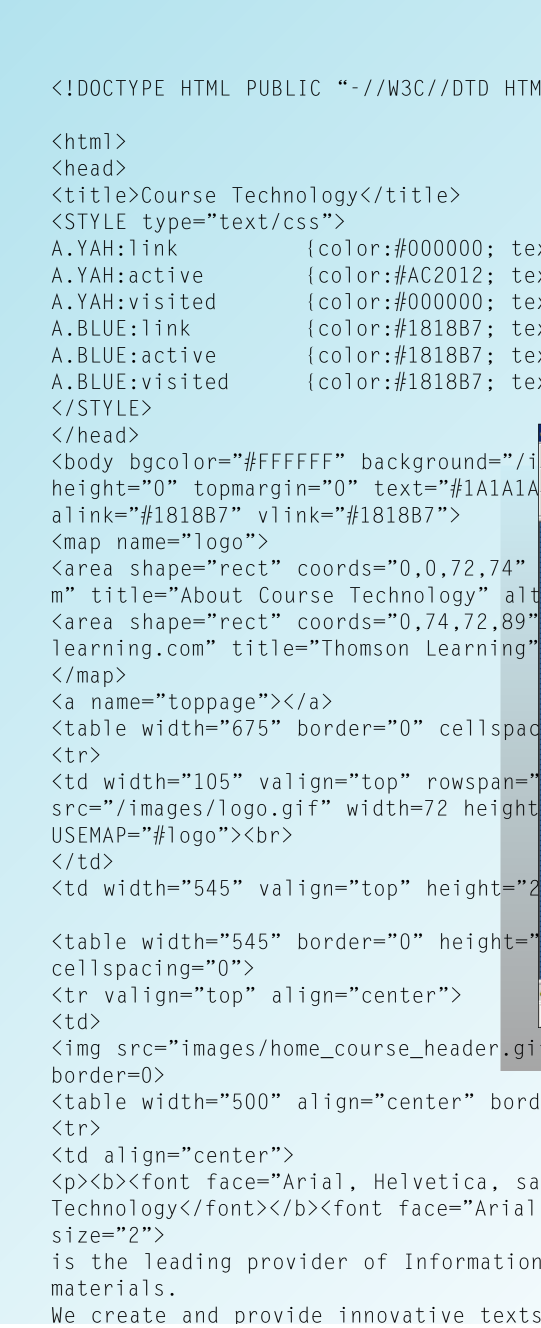
WEB PAGE PROGRAM DEVELOPMENT

As discussed in Chapter 2, the collection of linked documents accessible on the Internet is known as the World Wide Web, or simply, the Web. Each document posted on the Web, called a Web page, is a linked document that can contain text, graphics, video, and audio. The designers of Web pages, called **Web page authors**, use a variety of techniques to develop Web pages. The following sections discuss these techniques.

HTML

Hypertext markup language (**HTML**) is a special formatting language that programmers use to create Web pages. You view a Web page written with HTML in a Web browser such as Microsoft Internet Explorer or Netscape Navigator.

HTML is not actually a programming language. It is, however, a language that has specific syntax rules for defining the placement and format of text, graphics, video, and audio on a Web page. HTML uses **tags**, or **markups**, which are codes that specify links to other documents and indicate how the Web page displays when viewed on the Web. A Web page, thus, is a file that contains both text and HTML tags. Examples of tags are <B> to bold text, <P> to indicate a new paragraph, and <HR> to display a horizontal rule across the page. Figure 15-33 shows part of the HTML code used to create the Web page shown in Figure 15-34.

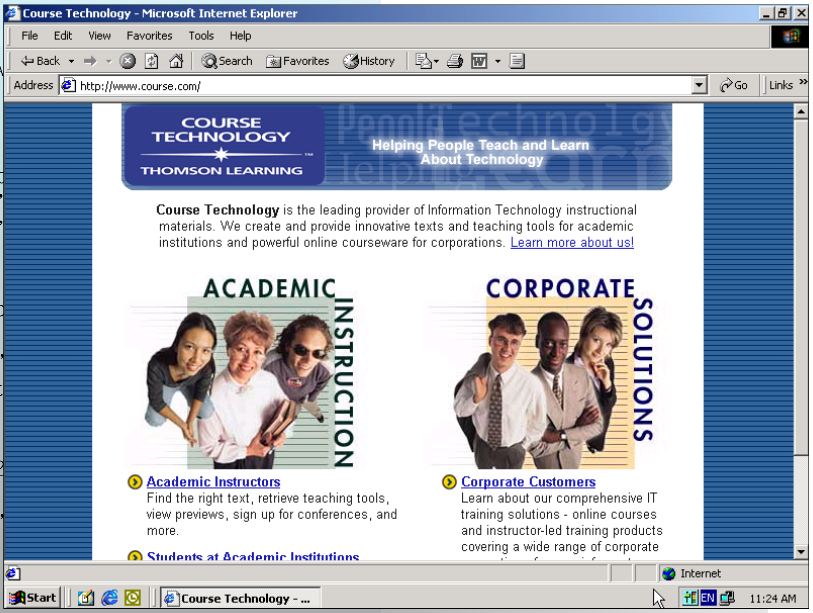


```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.0 Transitional//EN”>

<html>
<head>
<title>Course Technology</title>
<STYLE type=”text/css”>
A.YAH:link          {color:#000000; text-decoration:underline;}
A.YAH:active        {color:#AC2012; text-decoration:underline;}
A.YAH:visited       {color:#000000; text-decoration:underline;}
A.BLUE:link         {color:#1818B7; text-decoration:underline;}
A.BLUE:active       {color:#1818B7; text-decoration:underline;}
A.BLUE:visited      {color:#1818B7; text-decoration:underline;}
</STYLE>
</head>
<body bgcolor=”#FFFFFF” background=”/i
height=”0” topmargin=”0” text=”#1A1A1A
alink=”#1818B7” vlink=”#1818B7”>
<map name=”logo”>
<area shape=”rect” coords=”0,0,72,74”
m” title=”About Course Technology” alt
<area shape=”rect” coords=”0,74,72,89”
learning.com” title=”Thomson Learning”
</map>
<a name=”toppage”></a>
<table width=”675” border=”0” cellspac
<tr>
<td width=”105” valign=”top” rowspan=”
src=”/images/logo.gif” width=72 height
USEMAP=”#logo”><br>
</td>
<td width=”545” valign=”top” height=”2

<table width=”545” border=”0” height=”
cellspacing=”0”>
<tr valign=”top” align=”center”>
<td>
<img src=”images/home_course_header.gif” width=”545” height=”32”
border=0>
<table width=”500” align=”center” border=0>
<tr>
<td align=”center”>
<p><b><font face=”Arial, Helvetica, sans-serif” size=”2”>Course
Technology</font></b><font face=”Arial, Helvetica, sans-serif”
size=”2”>
is the leading provider of Information Technology instructional
materials.
We create and provide innovative texts and teaching tools for aca-
```

Figure 15-33 Hypertext Markup Language (HTML) is used to create Web pages. Illustrated is part of the HTML code that generates a portion of the Web page shown in Figure 15-34.


Course Technology - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://www.course.com/
Go Links
COURSE TECHNOLOGY
THOMSON LEARNING
Helping People Teach and Learn About Technology
ACADEMIC INSTRUCTION
CORPORATE SOLUTIONS
Internet
Start
Course Technology - ...
EN
11:24 AM

Course Technology is the leading provider of Information Technology instructional materials. We create and provide innovative texts and teaching tools for academic institutions and powerful online courseware for corporations. Learn more about us!

Academic Instructors

Find the right text, retrieve teaching tools, view previews, sign up for conferences, and more.

Students at Academic Institutions

Corporate Customers

Learn about our comprehensive IT training solutions - online courses and instructor-led training products covering a wide range of corporate

Figure 15-34 Web page created from part of the HTML code shown in Figure 15-33.

You can write HTML code using any text editor such as Notepad. The HTML code also can be entered into any standard word processing software package, such as Microsoft Word. You must save the code, however, as an ASCII file with an .htm or .html extension, instead of as a formatted word processing document.

Scripts, Applets, Servlets, and ActiveX Controls

HTML tells the browser how to display text and images, set up lists and option buttons, and establish hyperlinks on a Web page. These Web pages come to life when you add dynamic content and interactive elements such as scrolling messages, animated graphics, forms, pop-up windows, and interaction. To add these elements, you write small programs called scripts, applets, servlets, and ActiveX controls.

Scripts, applets, servlets, and ActiveX controls are short programs that run inside of another program. This is different from programs discussed thus far, which are executed by the operating system. In the case of Web pages, the Web browser executes these short programs.

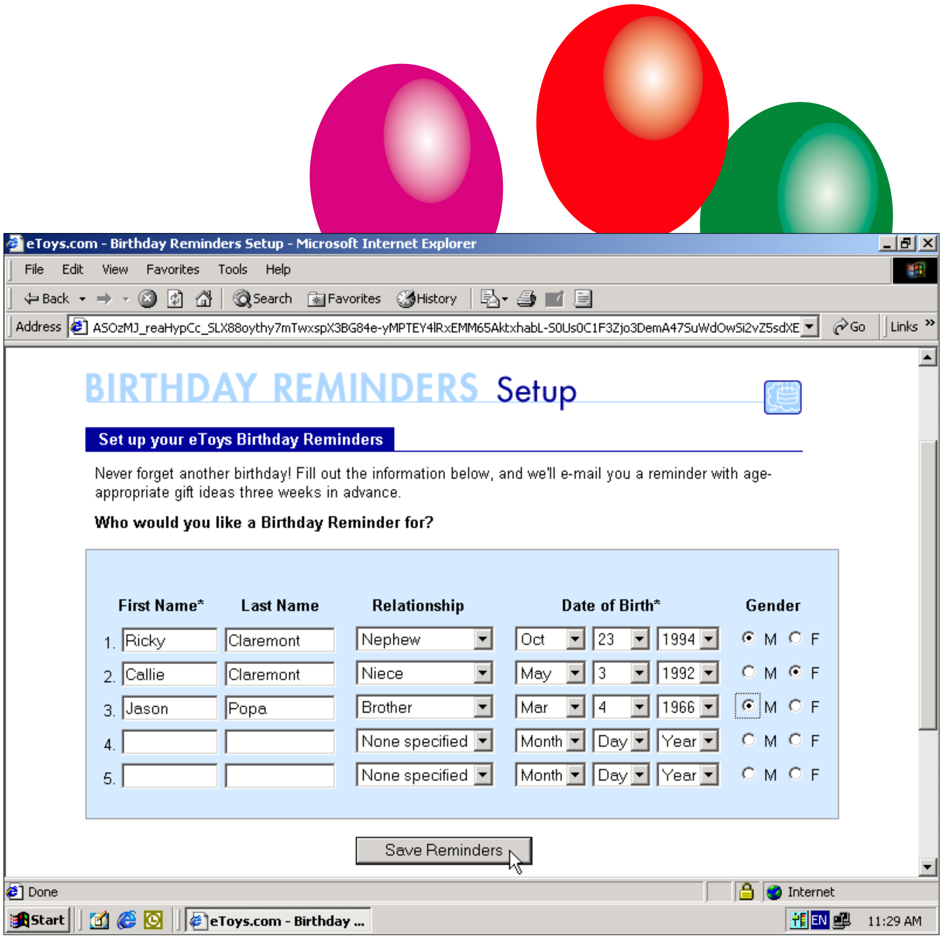
As discussed in Chapter 9, your computer is the client computer when it is connected to the Web. A **script** is an interpreted program that runs on the client. A script runs on your computer, instead of running on a Web server. An **applet** also usually runs on the client, but it is compiled. Thus, an applet usually runs faster than a script. Scripts and applets shift the computational work from the Web server to your computer because they run on the client. A **servlet** is an applet that runs on the server.

Similar to an applet, an **ActiveX control** is a small program that runs on your computer, instead of the server. ActiveX controls use ActiveX technology. **ActiveX** is a set of object-oriented technologies by Microsoft that allow components on a network to communicate with one another. To run an ActiveX control, your browser must support ActiveX technology. If it does not, you will need a plug-in program to run ActiveX controls.

One reason for using scripts, applets, servlets, and ActiveX controls is to add special multimedia effects to Web pages. Examples include animated graphics, scrolling messages, calendars, and advertisements. Another reason to use these programs is to include interactive capabilities on Web pages. Cookies, shopping carts, games, counters, image maps, and processing forms, are types of scripts, applets, servlets, and ActiveX controls that allow you to transfer information to and from a Web server.

A **counter** tracks the number of visitors to a Web site. An image map is a graphical image that points to a URL. Web pages use **image maps** in place of, or in addition to, plain text hyperlinks. When you click a certain part of the graphical image, your Web browser sends the coordinates of the clicked location to the Web server, which in turn locates the corresponding URL and sends the Web page to your computer.

A **processing form** collects data from visitors to a Web site, who fill in blank fields and then click a button that sends the information (Figure 15-35). The script or applet executes when a user clicks this


eTroys.com - Birthday Reminders Setup - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address
Go
Links
Done
Internet
Start
eToys.com - Birthday ...
EN
11:29 AM

Address: ASOzMJ_reaHypCc_SLX88oythy7mTwxspX3BG84e-yMPTEY4lRxEMM65AktxhabL-S0Us0C1F3Zjo3DemA47SuWdOwSi2vZ5sdXE

BIRTHDAY REMINDERS Setup

Set up your eToys Birthday Reminders

Never forget another birthday! Fill out the information below, and we'll e-mail you a reminder with age-appropriate gift ideas three weeks in advance.

Who would you like a Birthday Reminder for?

	First Name*	Last Name	Relationship	Date of Birth*			Gender
1.	Ricky	Claremont	Nephew	Oct	23	1994	● M ○ F
2.	Callie	Claremont	Niece	May	3	1992	○ M ● F
3.	Jason	Popa	Brother	Mar	4	1966	● M ○ F
4.			None specified	Month	Day	Year	○ M ○ F
5.			None specified	Month	Day	Year	○ M ○ F

Save Reminders

Figure 15-35 One popular use of a script is to create a processing form. This processing form from a Web page at eToys asks for someone's first and last name, relationship to you, and birthdate so you can receive a birthday reminder message from eToys.

button on the processing form. It transmits the data to the server, processes it, and then, if appropriate, sends information back to your Web browser via the server.

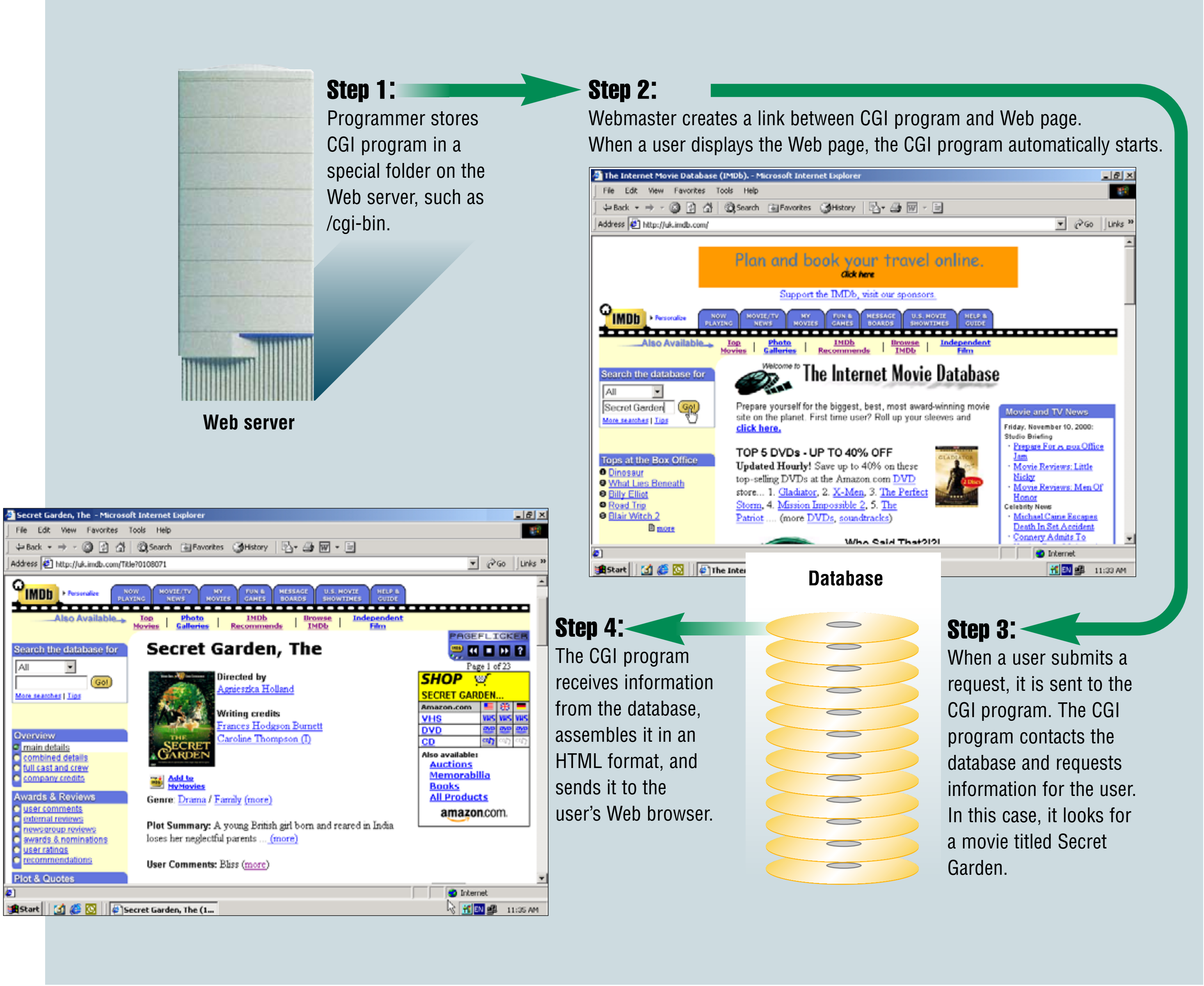
To send and receive information between your computer and a Web server, the script, applet, or servlet uses the common gateway interface. The **common gateway interface** (**CGI**) is the communications standard that defines how a Web server communicates with outside sources. Many times, the outside source is a database. The program that manages the sending and receiving across the CGI is a **CGI script**, also called a **CGI program**. The steps in Figure 15-36 show how a CGI program works.

A CGI program can be in the form of a script, applet, servlet, or ActiveX control. You can download CGI programs from the Web and purchase them. If one does not exist that meets your needs, you can write your own CGI program.

Web Link

For more information on CGI programs, visit the Discovering Computers 2002 Chapter 15 WEB LINK page (**scsite.com/dc2002/ch15/weblink.htm**) and click CGI Programs.

Figure 15-36 HOW A CGI PROGRAM WORKS



Step 1: Programmer stores CGI program in a special folder on the Web server, such as /cgi-bin.

Step 2: Webmaster creates a link between CGI program and Web page. When a user displays the Web page, the CGI program automatically starts.

Step 3: When a user submits a request, it is sent to the CGI program. The CGI program contacts the database and requests information for the user. In this case, it looks for a movie titled Secret Garden.

Step 4: The CGI program receives information from the database, assembles it in an HTML format, and sends it to the user's Web browser.

Java, JavaScript, VBScript, and Perl

You can write a CGI program using a variety of languages. These include some of the languages previously discussed, such as C, C++, and Visual Basic. Many programmers choose to use scripting languages to write CGI scripts. A **scripting language** is an interpreted language that typically is easy to learn and use. Popular scripting languages include JavaScript, VBScript, and Perl. Programmers often use Java for applets and servlets. The following paragraphs discuss each of these languages.

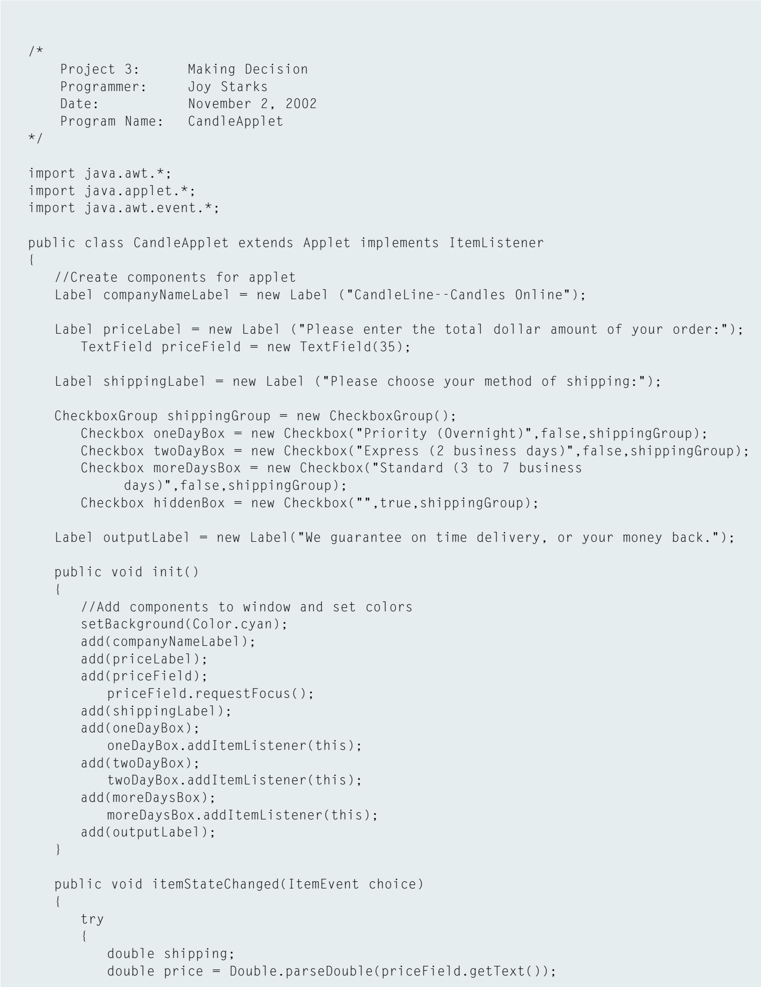
JAVA Developed by Sun Microsystems, **Java** is a compiled object-oriented programming language used to write stand-alone applications, as well as applets and servlets. Java applet examples might include input forms, rotating images, fireworks, interactive animations, or a game. Figure 15-37 shows a sample Java program and its resulting screen.

The Java language is very similar to C++. One difference is that Java source code is compiled into **bytecode**, instead of object code. The operating system cannot execute bytecode. A Java interpreter executes the bytecode. Java-enabled Web browsers contain Java bytecode interpreters.

Web Link

For more information on Java, visit the Discovering Computers 2002 Chapter 15 WEB LINK page (**scsite.com/dc2002/ch15/weblink.htm**) and click Java.

Figure 15-37a (Java applet code)



```
/*
    Project 3:      Making Decision
    Programmer:     Joy Starks
    Date:           November 2, 2002
    Program Name:   CandleApplet
*/

import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class CandleApplet extends Applet implements ItemListener
{
   //Create components for applet
   Label companyNameLabel = new Label ("CandleLine--Candles Online");

   Label priceLabel = new Label ("Please enter the total dollar amount of your order:");
      TextField priceField = new TextField(35);

   Label shippingLabel = new Label ("Please choose your method of shipping:");

   CheckboxGroup shippingGroup = new CheckboxGroup();
      Checkbox oneDayBox = new Checkbox("Priority (Overnight)",false,shippingGroup);
      Checkbox twoDayBox = new Checkbox("Express (2 business days)",false,shippingGroup);
      Checkbox moreDaysBox = new Checkbox("Standard (3 to 7 business
           days)",false,shippingGroup);
      Checkbox hiddenBox = new Checkbox("",true,shippingGroup);

   Label outputLabel = new Label("We guarantee on time delivery, or your money back.");

   public void init()
   {
      //Add components to window and set colors
      setBackground(Color.cyan);
      add(companyNameLabel);
      add(priceLabel);
      add(priceField);
         priceField.requestFocus();
      add(shippingLabel);
      add(oneDayBox);
         oneDayBox.addItemListener(this);
      add(twoDayBox);
         twoDayBox.addItemListener(this);
      add(moreDaysBox);
         moreDaysBox.addItemListener(this);
      add(outputLabel);
   }

   public void itemStateChanged(ItemEvent choice)
   {
      try
      {
         double shipping;
         double price = Double.parseDouble(priceField.getText());
```

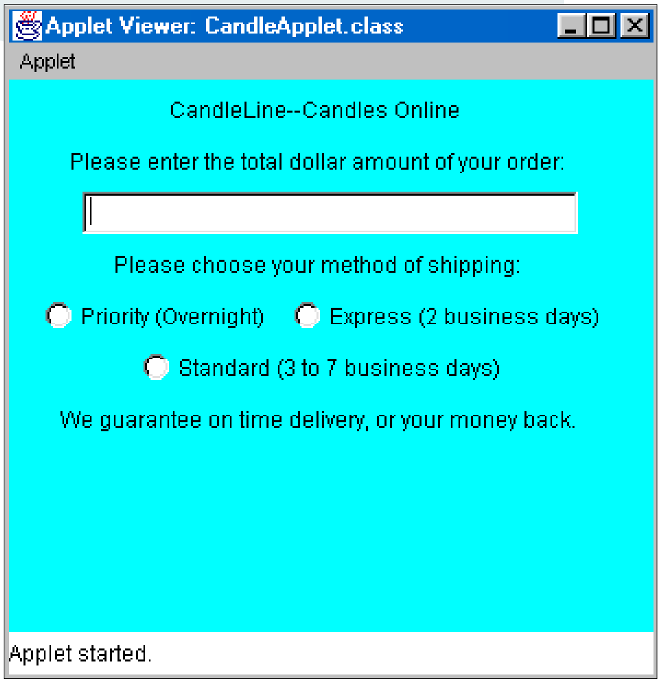

Applet Viewer: CandleApplet.class
Applet
CandleLine--Candles Online
Please enter the total dollar amount of your order:
Please choose your method of shipping:
○ Priority (Overnight)
○ Express (2 business days)
○ Standard (3 to 7 business days)
We guarantee on time delivery, or your money back.
Applet started.

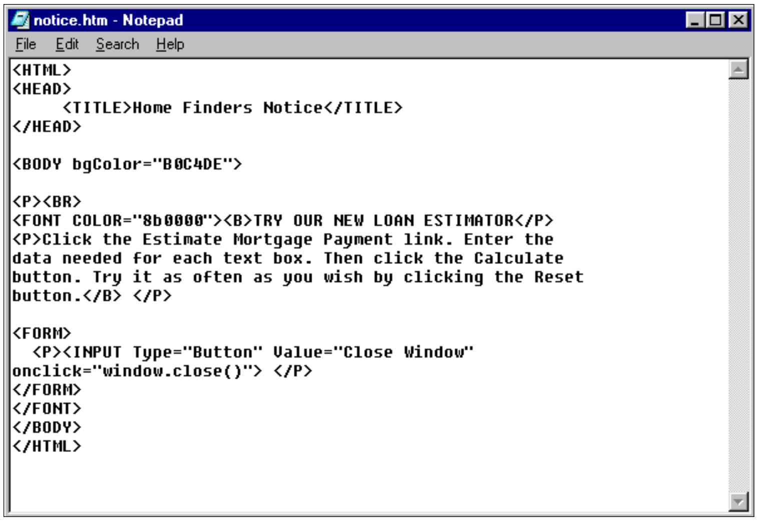
Figure 15-37b (resulting screen)

Figure 15-37 Sample Java applet code and the screen it displays.

Code segments used to create a Java application are called **JavaBeans**, or **Beans**. A JavaBean is platform independent. This enables the code to run on any computer or operating system. Many programmers believe that Java will be the programming language of the future because of its simplicity, robustness, and portability.

JAVASCRIPT **JavaScript** is an interpreted language that allows the programmer to add dynamic content and interactive elements to a Web page (Figure 15-38). These elements include alert messages, scrolling text, animations, drop-down menus, data input forms, pop-up windows, and interactive quizzes. You insert JavaScript code directly into an HTML document. Although it shares many of the features of the full Java language, JavaScript is a much simpler language.

Figure 15-38a (JavaScript code)



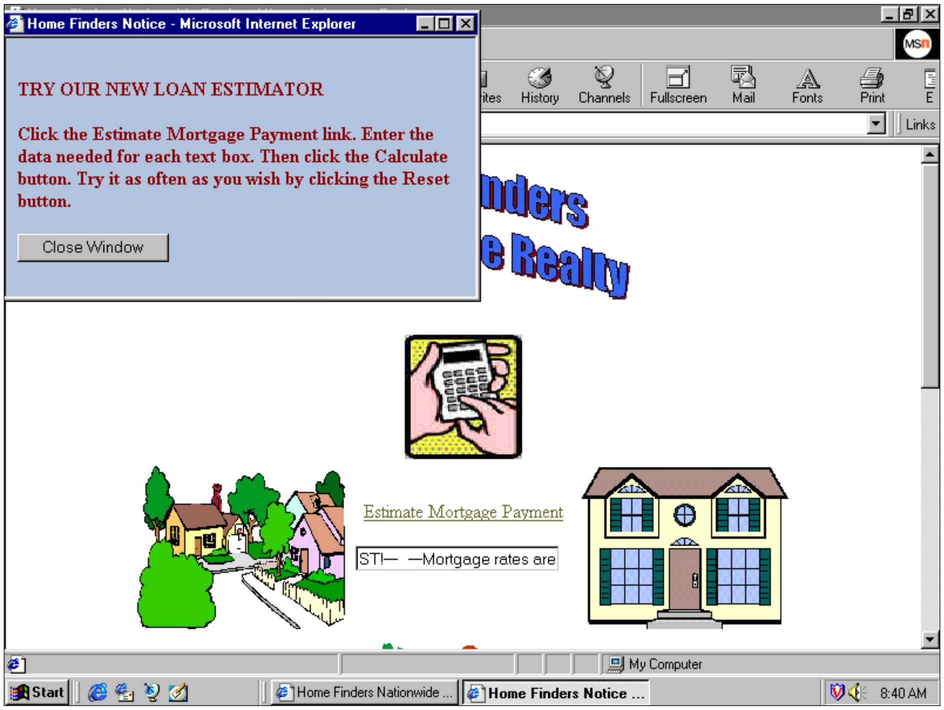
```
<HTML>
<HEAD>
     <TITLE>Home Finders Notice</TITLE>
</HEAD>

<BODY bgColor="B0C4DE">

<P><BR>
<FONT COLOR="8b0000"><B>TRY OUR NEW LOAN ESTIMATOR</P>
<P>Click the Estimate Mortgage Payment link. Enter the
data needed for each text box. Then click the Calculate
button. Try it as often as you wish by clicking the Reset
button.</B> </P>

<FORM>
  <P><INPUT Type="Button" Value="Close Window"
onclick="window.close()"> </P>
</FORM>
</FONT>
</BODY>
</HTML>
```

Figure 15-38b (pop-up window)



TRY OUR NEW LOAN ESTIMATOR

Click the Estimate Mortgage Payment link. Enter the data needed for each text box. Then click the Calculate button. Try it as often as you wish by clicking the Reset button.

Close Window

Figure 15-38 Shown here is a pop-up window that displays on a Web page and its associated JavaScript code.

ISSUE

100% Pure

Standardizing Java

"100% Pure Java" may soon be as familiar on software packages as it is on coffee shop menus. Java is called the programming language of the future. A simple language, Java utilizes an object-oriented approach, module orientation, and an easy-to-use GUI. In addition, Java offers a cross-platform capability that allows it to be used in any computing environment. Some software companies, however, are releasing nonstandard Java applications. This contaminated Java, which incorporates certain proprietary elements, runs only on specific computers. A number of vendors want standard Java applications to be awarded a "100% Pure Java" logo. Do you think this is a good idea? Why or why not? Who would test, and confirm, the purity of Java applications? Who might oppose this certification? Why?

For more information on Java, visit the Discovering Computers 2002 Issues Web page (**scsite.com/dc2002/issues.htm**) and click Chapter 15 Issue #5.

JavaScript is the result of a joint venture between Sun Microsystems and Netscape Communications Corporation. Netscape originally began developing a scripting language called LiveScript. Simultaneously, Sun tried to simplify its Java programming language. Today, many software companies endorse JavaScript. The program is an **open language**, which means anyone can use it without purchasing a license. JavaScript thus allows the programmer to improve the appearance of Web pages without spending a large amount of money.

To run a JavaScript program, your browser must support it. Netscape supports it. Internet Explorer supports a subset of JavaScript, called **JScript**.

Web Link

For more information on JavaScript, visit the Discovering Computers 2002 Chapter 15 WEB LINK page (**scsite.com/dc2002/ch15/weblink.htm**) and click JavaScript.

APPLY IT!

JavaScript Magic

JavaScript is a scripting language that lets you create applications that run over the Internet. A scripting language is like a macro — it is a list of commands that can be executed without user interaction. JavaScript is an open language anyone can use without purchasing a license. Although Netscape developed JavaScript, Microsoft Internet Explorer and other browsers support it. If you are developing Web pages and wish to include dynamic content, then consider adding JavaScript.

Compared with programming languages like Java, Visual Basic, and C, JavaScript is fairly simple to learn and use. Other options are available if, however, you do not have the time or inclination to devote to learning this scripting language. The Internet contains a wealth of resources. You can find example code, free tutorials, references, and even thousands of free scripts. For instance, you easily can create Web pages that include games such as checkers, interactive jigsaw puzzles, crosswords, and others. Or perhaps, you want to let users choose their own background color or add banners or a guest book. You can accomplish all of this, and much more, by cutting and pasting these free scripts into your Web page.

For more information on JavaScript, visit the Discovering Computers 2002 Apply It Web page (**scsite.com/dc2002/apply.htm**) and click Chapter 15 Apply It #3.

VBSCRIPT **Visual Basic Scripting Edition**, also called **VBScript**, is a subset of the Visual Basic language that allows you to add intelligence and interactivity to Web pages. As with JavaScript, you embed VBScript code directly into an HTML document. Programmers already familiar with Visual Basic choose VBScript as their scripting language, so they do not have to learn a new scripting language. The latest version of Internet Explorer includes VBScript.

PERL **Perl** (**P**ractical **E**xtraction and **R**eport **L**anguage) originally was developed by Larry Wall at NASA's Jet Propulsion Laboratory as a procedural language similar to C. The latest release of PERL, however, is an interpreted scripting language. PERL has powerful text processing capabilities, enabling it to become a popular language for writing scripts.

Dynamic HTML

Dynamic HTML (**DHTML**) is a newer type of HTML that allows you to include more graphical interest and interactivity in a Web page, without the Web page accessing the Web server. When a programmer uses DHTML to create Web pages, the client's computer automatically can update and change its own content. These Web pages display much faster than Web pages created with basic HTML.

Typically, Web pages created with DHTML are much more animated and responsive to user interaction. Once these pages have downloaded, the page seems to come to life. Colors change, font sizes grow, objects appear and disappear as you move the mouse, and animations dance around the screen. Figure 15-39 shows an example of a Web page containing DHTML.

Dynamic HTML works by using the document object model, style sheets, and scripting languages. The **document object model** (**DOM**) defines every item on a Web page as an object. Fonts, graphics, headlines, tables, and every other page element are objects. With DOM, you can change properties such as color or size, of any or all of these objects on the Web page.

A style sheet contains descriptions of a document's characteristics. Many word processing documents use style sheets to define formats of characters and paragraphs. **Cascading style sheets** (**CSS**) contain the formats for how a particular object should display in a Web browser. For example, the CSS specify items such as background colors and images and link colors, fonts, and font sizes. A single HTML document can contain multiple cascading style sheets, thus, the name cascading. As a user moves the mouse or clicks an item, a new style sheet can be applied to change the appearance of the screen.

Once you have defined and formatted objects on a Web page, a scripting language manipulates them. A script can move, display, hide, or change the appearance of an object as the mouse moves over the object.

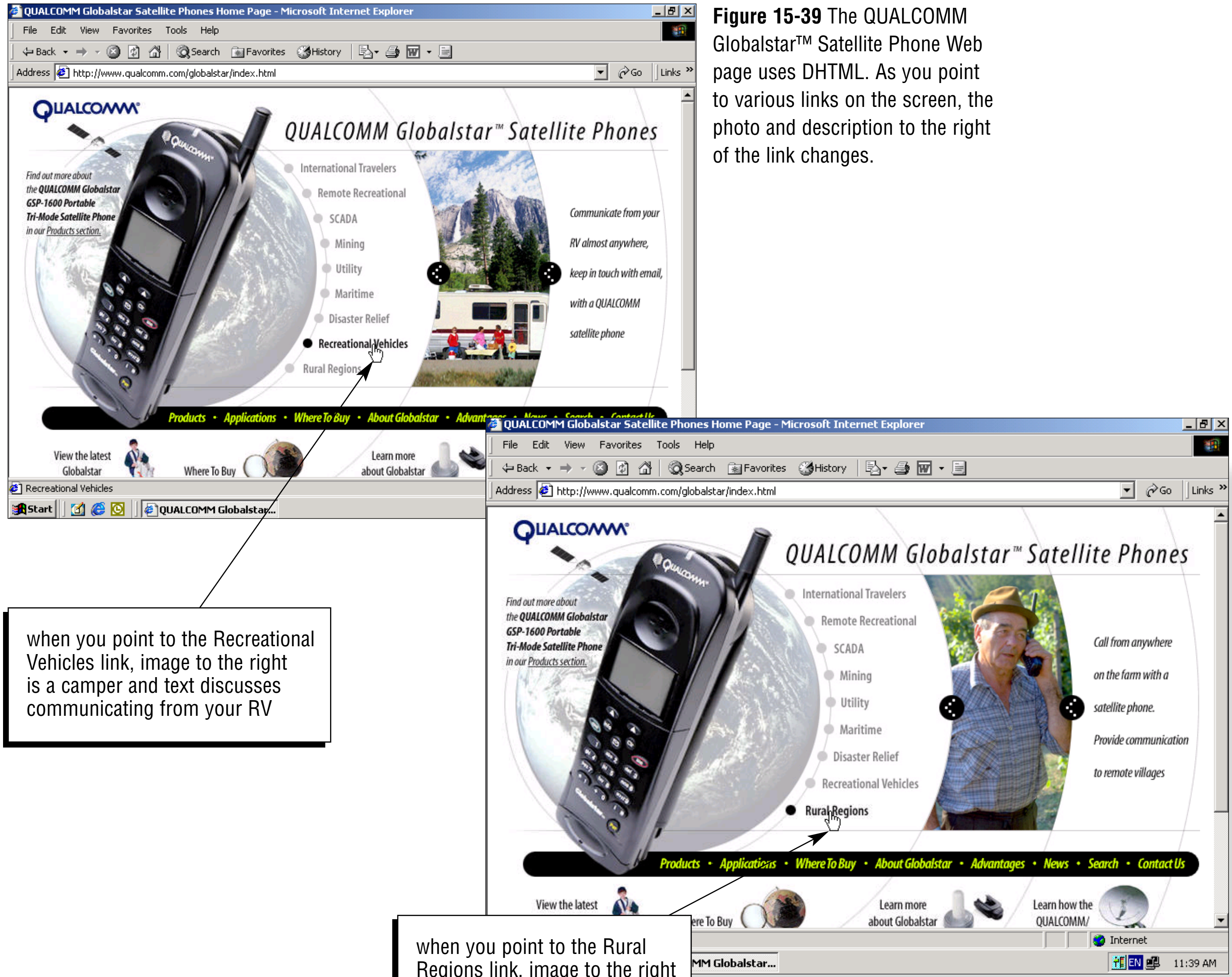
XHTML, XML, and WML

XHTML (eXtensible HTML) includes features of HTML and XML. **XML (eXtensible Markup Language)** allows Web page developers to create customized tags, as well as use predefined tags. With XML, you can define a link that points to multiple Web sites instead of a single site. XML uses **XSL (eXtensible Stylesheet Language)** as its style sheet specification.

XML separates the Web page content from its format, allowing your browser to display Web page contents in a form appropriate for your display device. For example, a handheld computer, a laptop computer, and a desktop computer all could display the same XML page. A Web page written with only HTML probably would require multiple versions to run on each of these types of computers.

Web Link

For more information on dynamic HTML, visit the Discovering Computers 2002 Chapter 15 WEB LINK page (**scsite.com/dc2002/ch15/weblink.htm**) and click Dynamic HTML.



QUALCOMM Globalstar Satellite Phones Home Page - Microsoft Internet Explorer

Address http://www.qualcomm.com/globalstar/index.html

Find out more about the QUALCOMM Globalstar GSP-1600 Portable Tri-Mode Satellite Phone in our Products section.

Communicate from your RV almost anywhere, keep in touch with email, with a QUALCOMM satellite phone

Call from anywhere on the farm with a satellite phone. Provide communication to remote villages

File Edit View Favorites Tools Help
Back Search Favorites History
Go Links
QUALCOMM
QUALCOMM Globalstar™ Satellite Phones
International Travelers
Remote Recreational
SCADA
Mining
Utility
Maritime
Disaster Relief
Recreational Vehicles
Rural Regions
Products • Applications • Where To Buy • About Globalstar • Advantages • News • Search • Contact Us
View the latest Globalstar
Where To Buy
Learn more about Globalstar
Learn how the QUALCOMM/
Recreational Vehicles
Internet
Start
QUALCOMM Globalstar...
11:39 AM

when you point to the Recreational Vehicles link, image to the right is a camper and text discusses communicating from your RV

when you point to the Rural Regions link, image to the right is a farmer and text discusses calling from the farm

Figure 15-39 The QUALCOMM Globalstar™ Satellite Phone Web page uses DHTML. As you point to various links on the screen, the photo and description to the right of the link changes.

Wireless devices use a subset of XML called WML. **WML** (**wireless markup language**) allows Web page developers to design pages specifically for microbrowsers. Many Web-enabled handheld computers, PDAs, cellular phones, and pagers use WML as their markup language. WML uses the **wireless application protocol** (**WAP**), a standard that specifies how wireless devices communicate with the Web.

Web Link

For more information on WML, visit the Discovering Computers 2002 Chapter 15 WEB LINK page (**scsite.com/dc2002/ch15/weblink.htm**) and click WML.

Before using XHTML, DHTML, or WML, be sure your browser supports them. The World Wide Web Consortium (W3C) is developing standards for new HTML features.

Web Page Authoring Software

As discussed in Chapter 3, you do not need to learn HTML to develop a Web page. Both new and experienced users can create fascinating Web sites with Web page authoring software. **Web page authoring software**, sometimes called an **HTML editor**, allows you to create sophisticated Web pages that include graphical images, video, audio, animation, and other special effects. Popular Web page authoring packages include Adobe GoLive, Lotus FastSite, Macromedia Dreamweaver, Macromedia Flash, and Microsoft FrontPage.

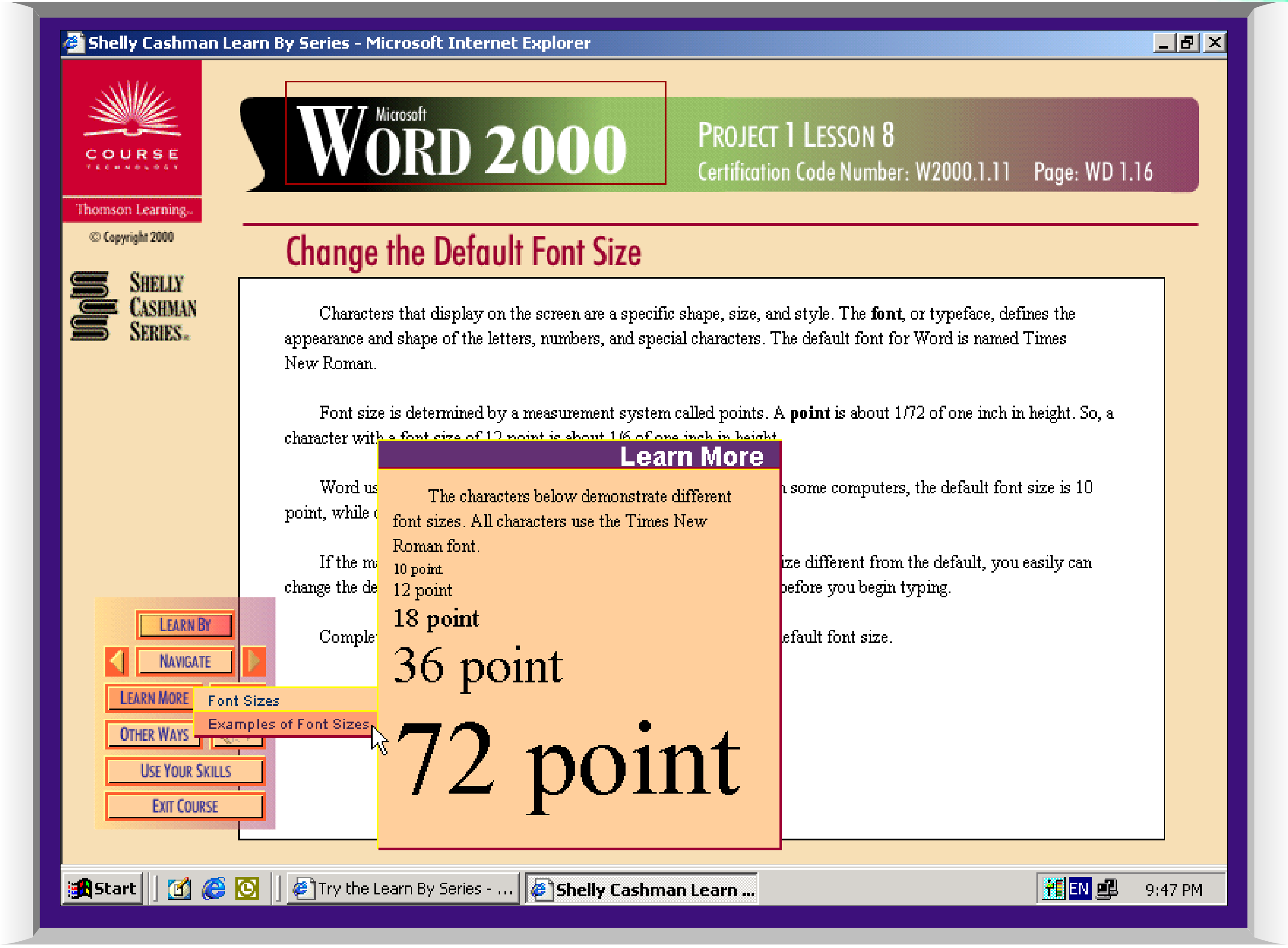
Web page authoring software generates HTML tags from your Web page design. With the Web page authoring software, you can view or modify the HTML associated with a Web page. Sometimes you may add an HTML tag that the Web page authoring software does not provide. The benefits to learning HTML basics enable you to fine-tune Web page formats created with authoring software.

Many application software packages also include Web page authoring features. With packages such as Microsoft Word or Excel, you can create basic Web pages that contain text and graphical images. Instead of or along with Web page authoring software, many Web developers use multimedia authoring software. This software allows developers to add additional special multimedia effects to Web pages. The next section discusses multimedia authoring software.

MULTIMEDIA PROGRAM DEVELOPMENT

Multimedia authoring software, sometimes called **authorware**, allows you to combine text, graphics, animation, audio, and video into an interactive presentation. Many developers use multimedia authoring software for computer-based and Web-based training environments.

As discussed in the Multimedia Special Feature in this book, **computer-based training** (**CBT**) is a type of education in which students learn by using and completing exercises with instructional software



Change the Default Font Size

Characters that display on the screen are a specific shape, size, and style. The **font**, or typeface, defines the appearance and shape of the letters, numbers, and special characters. The default font for Word is named Times New Roman.

Font size is determined by a measurement system called points. A **point** is about 1/72 of one inch in height. So, a character with a font size of 12 point is about 1/6 of one inch in height.

Word us ... some computers, the default font size is 10 point, while ...

If the m ... ize different from the default, you easily can change the de ... before you begin typing.

Comple ... efault font size.

The characters below demonstrate different font sizes. All characters use the Times New Roman font.

10 point

12 point

18 point

36 point

72 point

Figure 15-40 Computer-based training such as the Learn By Series is designed so students can choose learning activities that complement their learning styles. Five distinctive Learn By activities and a simulated environment where students use their skills guarantee an effective learning experience.

(Figure 15-40). Also called **computer-aided instruction** (**CAI**), this type of training is popular in business, industry, and schools to teach new skills or enhance the existing skills of employees, teachers, or students. Interactive CBT software called **courseware** usually is available on CD-ROM or DVD-ROM or shared over a network.

Web-based training (**WBT**) is an approach to computer-based training (CBT) that uses the technologies of the Internet and the Web. Today, many major companies provide employees with some type of WBT to teach new skills or upgrade their current skills.

WBT, CBT, and other materials often are combined as materials for distance learning courses. **Distance learning**, also called **distance education** or **online learning**, is the delivery of education at one location while the learning takes place at other locations. Many national and international companies, as well as colleges and universities, offer distance-learning training and courses.

Toolbook, Authorware, Director, and Flash

You can buy many CBT and WBT programs from a local retailer or merchant on the Web. You also can write them yourself using multimedia authoring software. Most of today's popular authoring packages share similar features and are capable of creating similar applications. Popular packages include Toolbook, Authorware, and Director.

TOOLBOOK **ToolBook**, from click2learn.com, Inc., has a graphical user interface and uses an object-oriented approach so you can design your applications using basic objects. These objects include buttons, fields, graphics, backgrounds, and pages.

In ToolBook, you can convert your multimedia application into HTML and Java so it can be distributed over the Internet. Figure 15-41 shows a sample application developed in ToolBook that generated DHTML. Many businesses and colleges use ToolBook to create content for distance learning courses.

AUTHORWARE **Authorware**, from Macromedia, is a multimedia authoring software package that provides the tools developers need to build interactive multimedia training and educational programs. Authorware offers a powerful authoring environment for the development of interactive multimedia magazines, catalogs, reference titles for CD-ROMs and DVD-ROMs, and applications for kiosks. Authorware also offers tools for bundling the content, student tracking, and course management components of a WBT or other distance learning course. You can view Authorware applications distributed over the Web using the Shockwave plug-in.

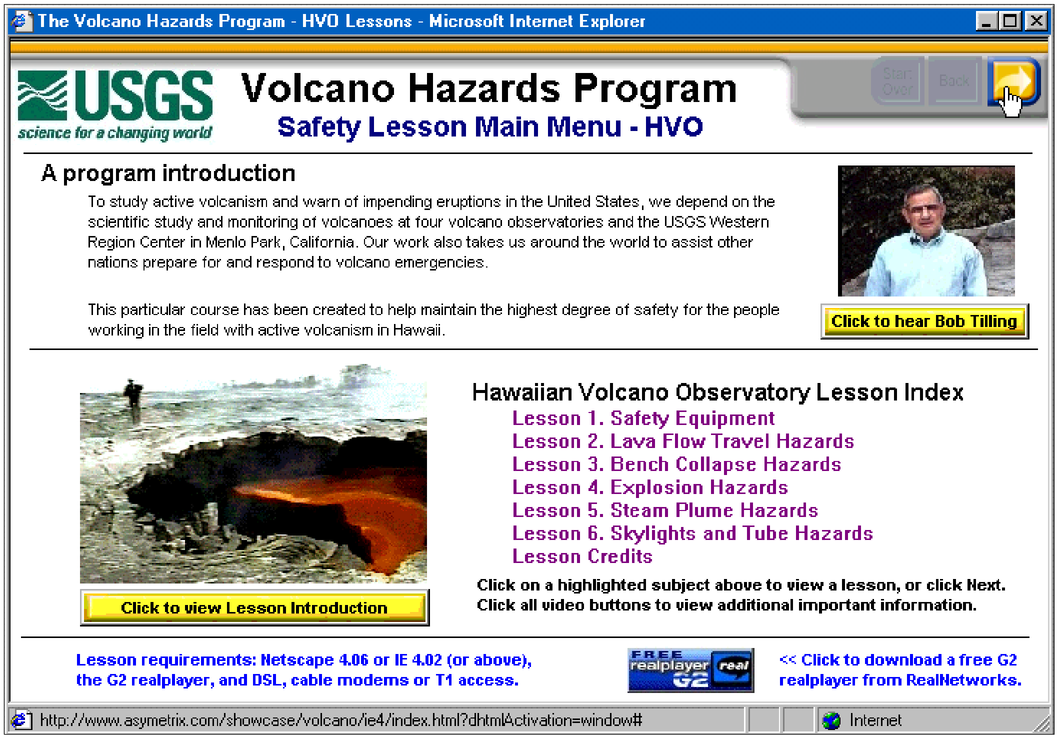
DIRECTOR **Director**, also from Macromedia, is a popular multimedia authoring program with powerful features that allow you to create highly interactive multimedia applications. The CBT application shown in Figure 15-40 on the previous page illustrates a multimedia application created in Director.

Director includes Lingo, a built-in object-oriented scripting language. Director's powerful features make it well suited for developing electronic presentations, CD-ROMs or DVD-ROMs for education and entertainment, and simulations. Web-based applications can include streaming audio and video, interactivity, and multiuser functionality. As with Authorware, you view applications developed in Director on the Web using the Shockwave browser plug-in.

SELECTING A PROGRAMMING LANGUAGE OR PROGRAM DEVELOPMENT TOOL

Each programming language and program development tool has its own unique characteristics. Many, however, have similar characteristics — making it difficult to select one for a program development task. When deciding which to use, you should consider the following factors.

1. Standards of the organization. Many organizations have standards that require programmers to use a particular language or development tool for all applications.


The Volcano Hazards Program - HVO Lessons - Microsoft Internet Explorer
USGS
science for a changing world
Volcano Hazards Program
Safety Lesson Main Menu - HVO
Start Over
Back
A program introduction
Click to hear Bob Tilling
Hawaiian Volcano Observatory Lesson Index
Lesson 1. Safety Equipment
Lesson 2. Lava Flow Travel Hazards
Lesson 3. Bench Collapse Hazards
Lesson 4. Explosion Hazards
Lesson 5. Steam Plume Hazards
Lesson 6. Skylights and Tube Hazards
Lesson Credits
Click to view Lesson Introduction
FREE realplayer G2 real
http://www.asymetrix.com/showcase/volcano/ie4/index.html?dhtmlActivation=window#
Internet

To study active volcanism and warn of impending eruptions in the United States, we depend on the scientific study and monitoring of volcanoes at four volcano observatories and the USGS Western Region Center in Menlo Park, California. Our work also takes us around the world to assist other nations prepare for and respond to volcano emergencies.

This particular course has been created to help maintain the highest degree of safety for the people working in the field with active volcanism in Hawaii.

Click on a highlighted subject above to view a lesson, or click Next.
Click all video buttons to view additional important information.

Lesson requirements: Netscape 4.06 or IE 4.02 (or above), the G2 realplayer, and DSL, cable modems or T1 access.

<< Click to download a free G2 realplayer from RealNetworks.

Figure 15-41 A sample Toolbook application that uses DHTML.

2. Interface with other programs. If a program is to work with other programs, you should write it in the same language or with the same development tool as the other programs or a language or development tool compatible with the other programs.
3. Suitability of the language to the application. Most languages and development tools are designed to work with particular applications such as business applications or scientific applications.
4. Portability to other systems. If an application runs on multiple types of computers (hardware platforms) and operating systems (software platforms), select a language or development tool common to these platforms.

CHAPTER SUMMARY

This chapter discussed each step in the program development life cycle and presented the tools used to make this process efficient. This chapter also explained various programming languages and program development tools used to write and develop computer programs. Finally, it presented a variety of Web development and multimedia development tools.

COMPANY ON THE CUTTING EDGE

macromedia®

Multimedia Dominates Web Sites

Want to demo a Lexus sports sedan without leaving your house? Check the features on a Timex I-Control watch on your laptop computer? Create fun and educational science projects with your handheld computer? You can, with the help of Macromedia's lineup of splashy products, including Dreamweaver, Flash Player, and Shockwave.

More than 1 million professional Web developers use Macromedia software worldwide to author, produce, deliver, and analyze Web content that is rich in motion, sound, and graphics. They make the most of the high-quality, interactive user interfaces, which have been translated in eight languages. Dreamweaver is the leading platform in 70 percent of the market. More than 1.4 million copies of Flash Player are downloaded from Macromedia's Web site daily, making it the most widely Web-distributed software in history. Nearly 20 million users have registered their copies of Shockwave, with an average of 70,000 new users each day.

For more information on Macromedia, visit the Discovering Computers 2002 Companies Web page (**scsite.com/dc2002/companies.htm**) and click Macromedia.

Career Corner

Programmer

If you are curious, creative, and enjoy problem solving and detailed work, then consider a career in programming. Programmers write the code that tells computers what to do. Most programmers specialize in one of three fields: systems programming, application programming, and Web development programming.

Some jobs may require that the programmer develop the entire program, while others require program maintenance. Likewise, you may work for a small company where you are responsible for the entire System Development Life Cycle or you may work for a larger company where you are part of a team and the duties are very specialized.

Academic credentials are essential for success in this career. A good foundation in programming logic is the key to your success with any programming language. Some of the more popular languages to consider are Visual Basic, Java, C, and C++. Surveys indicate that average salaries for junior level programmers are about $40,000 and approximately $70,000 for senior programmers.

To learn more about a career in programming, visit the Discovering Computers 2002 Careers Web page (**scsite.com/dc2002/careers.htm**) and click Programmer.

E-CAREER

IN SEARCH OF THE PERFECT JOB

Web Helps Career Hunt

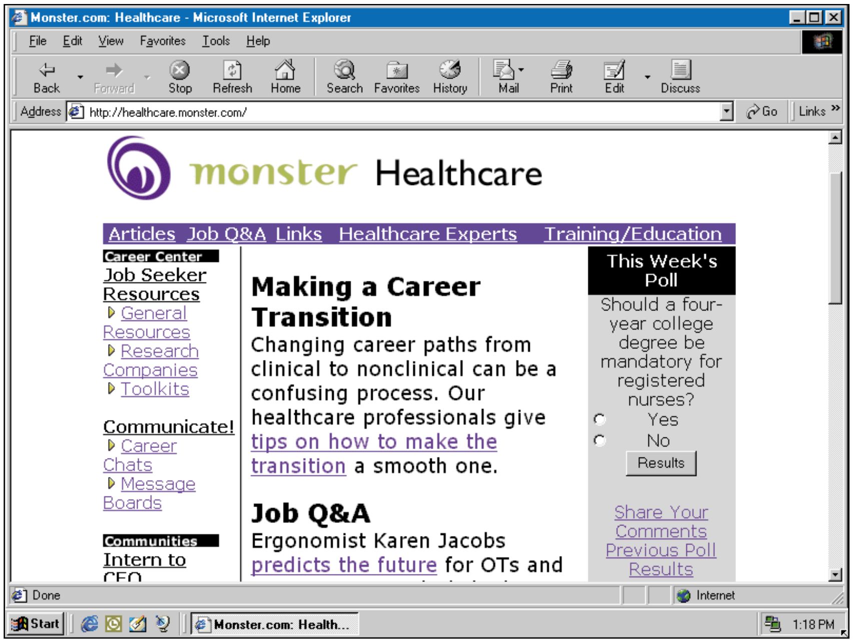
If you choose a job you love, you never will have to work a day in your life, according to Confucius. Does that sound good? All you need to do is find that perfect job, but that is no easy feat.

While your teachers give you valuable training to prepare you for a career, they rarely teach you how to begin that career. On-campus interviews are a start, but you can broaden your horizons by searching on the Internet for career information and job openings.

First, examine some of the job search Web sites. These resources list thousands of openings in hundreds of fields, companies, and locations. For example, the Monster Web site, shown in Figure 15-42, allows you to choose a broad job area, such as healthcare, then narrow your search to specific fields within that area, and then search for specific salary ranges, locations, and job functions. Other job search Web sites are listed in Figure 15-43.

Next, prepare your resume. Some Web sites have forms that allow you to type pertinent information into a blank form. Other Web sites want you to submit, or post a document. If so, write your resume using nouns to describe your skills and experience, such as team player, Spanish, and Microsoft Word. Be certain to spell check and scan the document for viruses. When you write your cover letter, use words the company incorporated in its job posting. Many companies use computer software applications, not humans, to find interviewees. The software searches for particular buzzwords, called keywords, in the files and then lists the documents that contain the most matches.

When a company contacts you for an interview, learn as much about it and the



Making a Career Transition
Changing career paths from clinical to nonclinical can be a confusing process. Our healthcare professionals give tips on how to make the transition a smooth one.

Job Q&A
Ergonomist Karen Jacobs predicts the future for OTs and

Should a four-year college degree be mandatory for registered nurses?
○ Yes
○ No

Figure 15-42 Monster.com's global online network connects companies with career-minded individuals.

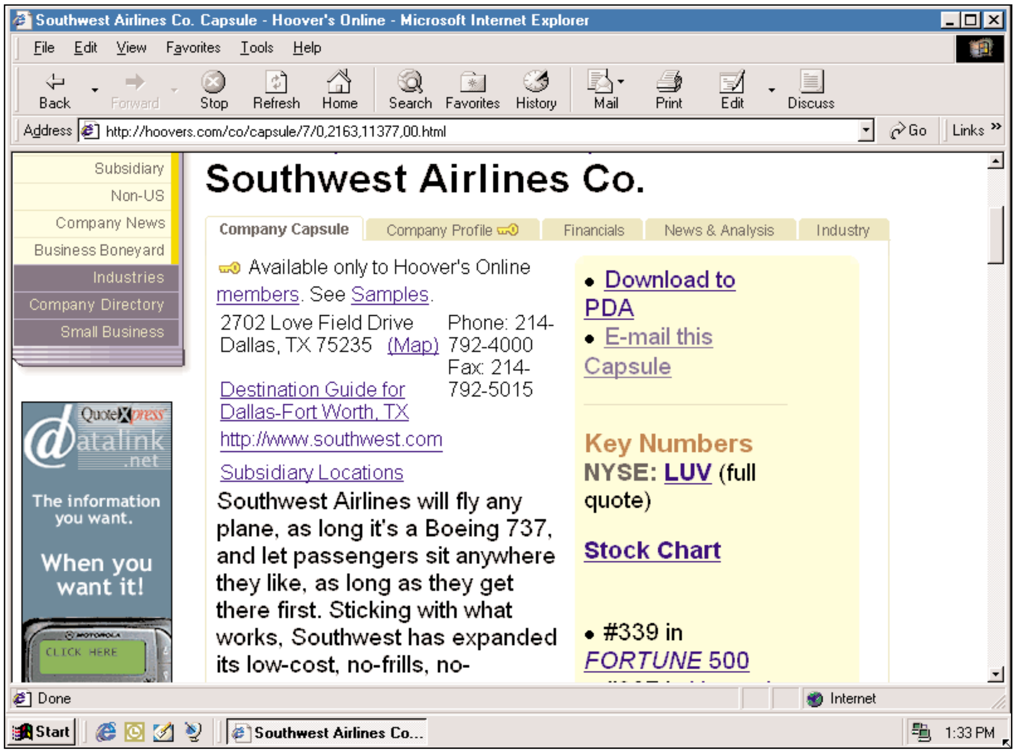
CAREER WEB SITES	URL
Job Search	
BestJobsUSA.com	bestjobsusa.com
Career Magazine	careermag.com
CareerBuilder	careerbuilder.com
CareerExchange.com	careerexchange.com
Careernet	careernet.com
College Grad Job Hunter	collegegrad.com
Headhunter.net	headhunter.net
HotJobs.com	hotjobs.com
JobBank USA	jobbankusa.com
JobOptions™	www.espan.com
JOBTRAK	jobtrak.com
JobWeb	www.jobweb.org
Monster Board	monster.com
The Employment Guide's CareerWeb	cweb.com
Company/Industry Information	
AmericanCompanies.com	www.americancompanies.com
Argus Clearinghouse	clearinghouse.net
Career ResourceCenter	www.resourcecenter.com
FORTUNE.com	fortune.com
Hoover's Online	hoovers.com
Occupational Outlook Handbook	stats.bls.gov/ocohome.htm

For an updated list of career Web sites, visit scsite.com/dc2002/e-rev.htm.

Figure 15-43 Career Web sites provide a variety of job openings and information on major companies worldwide.

industry as possible before the interview. Many of the Web sites listed in Figure 15-43 have detailed company profiles and links to their corporate Web sites. For instance, company information on Southwest Airlines is displayed in the Hoover's Online Web site, as shown in Figure 15-44. Also, look at Web sites for professional organizations and discussion groups for further insights.

For more information on using the Web for career information and for an updated list of career Web sites, visit the Discovering Computers 2002 E-Revolution Web page (scsite.com/dc2002/e-rev.htm) and click Careers.


Southwest Airlines Co. Capsule - Hoover's Online - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address http://hoovers.com/co/capsule/7/0,2163,11377,00.html
Go Links
Subsidiary
Non-US
Company News
Business Boneyard
Industries
Company Directory
Small Business
QuoteXpress
@atalink.net
The information you want.
When you want it!
MOTOROLA
CLICK HERE
Company Capsule
Company Profile
Financials
News & Analysis
Industry
Done
Internet
Start
Southwest Airlines Co...
1:33 PM

Southwest Airlines Co.

Available only to Hoover's Online members. See Samples.

2702 Love Field Drive
Dallas, TX 75235 (Map)

Phone: 214-792-4000
Fax: 214-792-5015

Destination Guide for Dallas-Fort Worth, TX

http://www.southwest.com

Subsidiary Locations

Southwest Airlines will fly any plane, as long it's a Boeing 737, and let passengers sit anywhere they like, as long as they get there first. Sticking with what works, Southwest has expanded its low-cost, no-frills, no-

- Download to PDA
- E-mail this Capsule

Key Numbers

NYSE: LUV (full quote)

Stock Chart

- #339 in FORTUNE 500

Figure 15-44 Company and industry information forms the core of Hoover's Online Web site.

e-REVOLUTION E-CAREER applied:

1. Use two of the job search Web sites listed in Figure 15-43 to find three companies with job openings in your field. Make a table listing the Web site name, position available, description, salary, location, desired education, and desired experience.

2. It is a good idea to acquire information prior to graduation about the industry in which you would like to work. Are you interested in the automotive manufacturing industrtoy, the restaurant service industry, or the financial industry? Use two of the company/industry information Web sites listed in Figure 15-43 to research a particular career related to your major. Write a paragraph naming the Web sites and the specific information you found, such as the nature of the work, recommended training and qualifications, employment outlook, and earnings. Then, use two other Web sites to profile three companies with positions available in this field. Write a paragraph about each of these companies, describing the headquarters' location, sales and earnings for the previous year, total number of employees, working conditions, perks, and competitors.

In Summary

Web Instructions: To display this page from the Web, start your browser and enter the URL scsite.com/dc2002/ch15/summary.htm. Click the links for current and additional information. To listen to an audio version of this In Summary, click the Audio button. To play the audio, RealPlayer must be installed on your computer (download by clicking here).

1

What Are the Six Steps in the Program Development Life Cycle?

The program development life cycle (PDLC) is a set of steps that programmers use to build a computer program. That program is the set of instructions that directs the computer to perform the steps necessary to process data into information. Step 1, analyze problem, consists of reviewing program specifications; meeting with the analyst and users; and identifying program components. Step 2, design programs, involves grouping activities into modules, devising **solution algorithms**, and testing the algorithms. Step 3, code programs, entails translating the solution algorithm into a programming language and entering program code into the computer. Step 4, test programs, consists of correcting **syntax errors** or **logic errors**. Step 5, formalize solution, includes reviewing program code and documentation. Step 6, maintain programs, involves correcting errors and adding enhancements.

2

What Is Top-Down Program Design?

Top-down design breaks the original set of program specifications into smaller, more manageable sections. A **module** is a section of program dedicated to performing a single function. Programmers use a hierarchy chart to graphically represent program modules. Programs developed using the top-down approach usually are reliable and easy to read and maintain.

3

What Are Structured Program Design and Control Structures?

Structured design is an approach wherein all program logic is constructed from a combination of three **control structures**, or designs, that direct the order in which program instructions are executed. A **sequence control structure** shows one or more actions following each other in sequence. A **selection control structure** tells the program which action to take based on a certain condition. Programmers use a repetition control structure when one or more actions are to be performed repeatedly as long as a certain condition is met.

4

How Are the Categories of Programming Languages Different?

A **programming language** is a set of words, symbols, and codes that enable a programmer to communicate a solution algorithm to the computer. **Machine language** uses a series of binary digits that correspond to the on and off electrical states of a computer. **Assembly language** instructions are written using abbreviations, codes, and **symbolic addresses**. **Third-generation language** (**3GL**) instructions use a series of English-like words. They are called procedural languages because the computer must be told what to accomplish and how to do it. A **fourth-generation language** (**4GL**), which also uses English-like statements, is a **nonprocedural language** because a programmer specifies only what the program should accomplish without explaining how. A **fifth-generation language** (**5GL**) provides a visual graphical interface for creating the source code. Machine and assembly languages are **low-level languages** written to run on one particular computer. Third-, fourth-, and fifth-generation languages are **high-level languages** that can run on various computers.

5

What Is the Object-Oriented Approach to Program Development?

With the **object-oriented (OO) approach**, a programmer can package the data and the procedure into a single unit called an **object**. The data elements in the object are called attributes, and the procedures are called **methods**. An **object-oriented programming** (**OOP**) **language** is used to implement the object-oriented approach to program development.

6 What Programming Languages Are Commonly Used Today?

BASIC is a simple, interactive problem-solving language sometimes used in an introductory programming course. **Visual Basic** is a programming language used to develop Windows-based applications. **COBOL** is a procedural programming language widely used for business applications. The **C** programming language is used to develop a variety of software, including operating systems and application programs. **C++** is an object-oriented extension of the C programming language. **RPG** is a nonprocedural language used for application development on IBM midrange computers.

7 Why Are Application Generators, Macros, and RAD Tools Used?

Program development tools such as application generators, macros, and RAD are user-friendly software products designed to create solutions to information systems requirements. An **application generator** is a program used to build an application without writing extensive code. A **macro** is a series of statements that instructs an application how to complete a routine, repetitive, or sometimes difficult task. **Rapid application development** (**RAD**) tools — such as **Visual Basic**, **Delphi**, and **PowerBuilder** — are used to develop software throughout the system development process instead of waiting until the implementation phase.

8 What Are the HTML, DHTML, XML, and WML Web Page Development Tools?

Hypertext markup language (HTML) is a special language used to create Web pages. It uses specific syntax rules for defining the placement and format of text, graphics, video, and sound on a Web page. **Dynamic HTML** (**DHTML**) is a newer type of HTML used to include more graphical interest and interactivity, without accessing the Web server. **eXtensible Markup Language** (**XML**) allows Web page developers to create customized **tags** (codes that specify links to other documents and indicate how the Web page displays), as well as use predefined tags. XML allows you to define a link that points to multiple Web sites. **WML** is a subset of XML and is used to design pages specifically for microbrowsers. **Scripts**, **applets**, and **servlets** are short programs that are executed inside of another program. The more common **scripting languages** are **VBScript, JavaScript**, and **Perl**. Applets and servlets commonly are written using **Java**.

9 Why are Multimedia Authoring Software Programs Used?

Multimedia authoring software allows you to combine text, graphics, animation, audio, and video into an interactive presentation. Applications created for **computer-based training** (**CBT**) and **Web-based training (WBT)** use this technology.

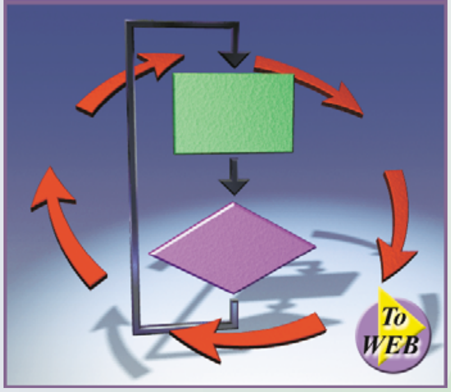
Key Terms

Web Instructions: To display this page from the Web, start your browser and enter the URL scsite.com/dc2002/ch15/terms.htm. Scroll through the list of terms. Click a term to display its definition and a picture. Click the To WEB button for current and additional information about the term from the Web. To see animations, Shockwave and Flash Player must be installed on your computer (download by clicking here).

ActiveX (15.32)
ActiveX control (15.32)
annotation symbol (15.10)
applet (15.32)
application generator (15.27)
assembler (15.17)
assembly language (15.16)
attributes (15.20)
authorware (15.38)
Authorware (15.40)
BASIC (15.22)
Beans (15.35)
bytecode (15.34)
C (15.25)
C# (15.25)
C++ (15.25)
cascading style sheets (CSS) (15.36)
case control structure (15.7)
CGI program (15.33)
CGI script (15.33)
class (15.20)
COBOL (15.24)
coding (15.13)
comment symbol (15.10)
comments (15.13)
common gateway interface (CGI) (15.33)
compiler (15.18)
component (15.29)
computer program (15.2)
computer-aided instruction (CAI) (15.39)
computer-based training (CBT) (15.38)
construct (15.6)
control structure (15.6)
CORBA (Common Object Request Broker Architecture) (15.21)

INFINITE LOOP

Set of instructions that repeats indefinitely, or forever. (15.8)



counter (15.32)
courseware (15.39)
dead code (15.8)
debug utility (15.14)
debugger (15.14)
debugging (15.14)
defining diagram (15.4)
Delphi (15.29)
design tools (15.9)
desk check (15.12)
Director (15.40)
distance education (15.39)
distance learning (15.39)
document object model (DOM) (15.36)
do-until control structure (15.8)
do-while control structure (15.8)
dynamic HTML (DHTML) (15.36)
empower (15.27)
encapsulation (15.20)
entry point (15.8)
event (15.21)
exit point (15.8)
fifth-generation language (5GL) (15.20)
flowchart (15.9)
flowcharting software (15.10)
fourth-generation language (4GL) (15.19)
global comments (15.13)
hierarchy chart (15.6)
high-level language (15.16)
HTML editor (15.38)
hypertext markup language (HTML) (15.31)
if-then-else control structure (15.7)
image maps (15.32)
implemented (15.15)
infinite loop (15.8)
information hiding (15.20)
inheritance (15.20)
internal comments (15.13)
interpreter (15.18)
IPO chart (15.4)
iteration control structure (15.8)
Java (15.34)
JavaBeans (15.35)
JavaScript (15.35)
JScript (15.36)
logic error (15.12)
loop (15.8)
low-level language (15.16)
machine language (15.16)
machine-dependent language (15.16)
machine-independent language (15.16)
macro (15.17, 15.28)
macro recorder (15.28)
maintaining (15.15)
markups (15.31)
menu generator (15.28)
message (15.21)
method (15.20)
millennium bug (15.14)
mnemonics (15.16)
module (15.5)
multimedia authoring software (15.38)
Nassi-Schneiderman (N-S) chart (15.11)
nonprocedural language (15.19)
object (15.20)
object code (15.18)
object instance (15.21)
Object Management Group (OMG) (15.21)
object program (15.18)
object-oriented (OO) approach (15.20)
object-oriented programming (OOP) language (15.21)
online learning (15.39)
open language (15.36)
operation (15.20)
Perl (15.36)
PowerBuilder (15.30)
procedural language (15.18)
processing form (15.32)
program development life cycle (PDLC) (15.2)
program development tool (15.16)
program enhancement (15.15)
program flowchart (15.9)
program generator (15.27)
program logic (15.8)
program specification package (15.2)
programming language (15.16)
programming team (15.4)
proper program (15.8)
pseudocode (15.12)
quality review (15.12)
query language (15.19)
rapid application development (RAD) (15.29)
remarks (15.13)
repetition control structure (15.8)
report generator (15.19)
report writer (15.19)
RPG (15.26)
run time errors (15.13)
script (15.32)
scripting language (15.34)
selection control structure (15.7)
sequence control structure (15.7)
servlet (15.32)
solution algorithm (15.8)
source program (15.17)
structure chart (15.6)
structured design (15.6)
structured flowcharts (15.12)
subclasses (15.20)
superclass (15.20)
symbolic address (15.16)
symbolic instruction codes (15.16)
syntax (15.13)
syntax error (15.13)
tags (15.31)
test data (15.12)
third-generation language (3GL) (15.18)
ToolBook (15.40)
top-down chart (15.6)
top-down design (15.5)
UML (Unified Modeling Language) (15.21)
variables (15.20)
VBScript (15.36)
Visual Basic Scripting Edition (15.36)
Visual Basic (15.22)
Visual Basic for Applications (VBA) (15.28)
visual programming environment (VPE) (15.29)
Web page authoring software (15.38)
Web page authors (15.31)
Web-based training (WBT) (15.39)
wireless application protocol (WAP) (15.38)
WML (wireless markup language) (15.38)
XHTML (eXtensible HTML) (15.37)
XML (eXtensible Markup Language) (15.37)
XSL (eXtensible Stylesheet Language) (15.37)
Y2K Bug (15.14)

Learn It Online

Web Instructions: To display this page from the Web, start your browser and enter the URL scsite.com/dc2002/ch15/learn.htm.

1. Web Guide

Click Web Guide to display the Guide to World Wide Web Sites and Searching Techniques page. Click Reference and then click Webopedia. Search for Visual Basic. Click one of the Visual Basic links. Use your word processing program to prepare a brief report on your findings and submit your assignment to your instructor.

2. Scavenger Hunt

Click Scavenger Hunt. Print a copy of the Scavenger Hunt page; use this page to write down your answers as you search the Web. Submit your completed page to your instructor.

3. Who Wants to Be a Computer Genius?

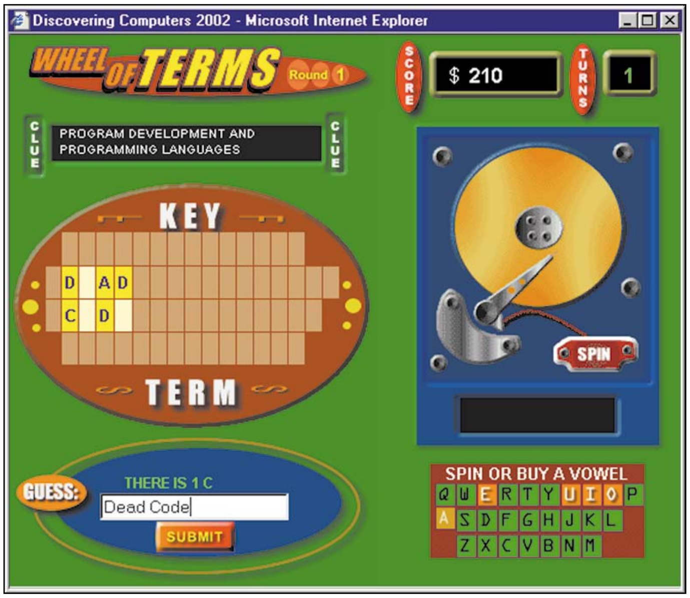
Click Computer Genius to find out if you are a computer genius. Directions on how to play the game will display. When you are ready to play, click the PLAY button. Submit your score to your instructor.

4. Wheel of Terms

Click Wheel of Terms to reinforce important terms you learned in this chapter by playing the Shelly Cashman Series version of this popular game. Directions on how to play the game will display. When you are ready to play, click the PLAY button. Submit your score to your instructor.

5. Career Corner

Click Career Corner to display the IBM page. Click one of the Facts and Information links and then review the information. Write a brief report describing what you learned. Submit the report to your instructor.



GUESS: Dead Code

6. Search Sleuth

Click Search Sleuth to learn search techniques that will help make you a research expert. Submit the completed assignment to your instructor.

7. Crossword Puzzle Challenge

Click Crossword Puzzle Challenge. Complete the puzzle to reinforce skills you learned in this chapter. Directions on how to play the game will display. When you are ready to play, click the PLAY button. Submit the completed puzzle to your instructor.

8. Practice Test

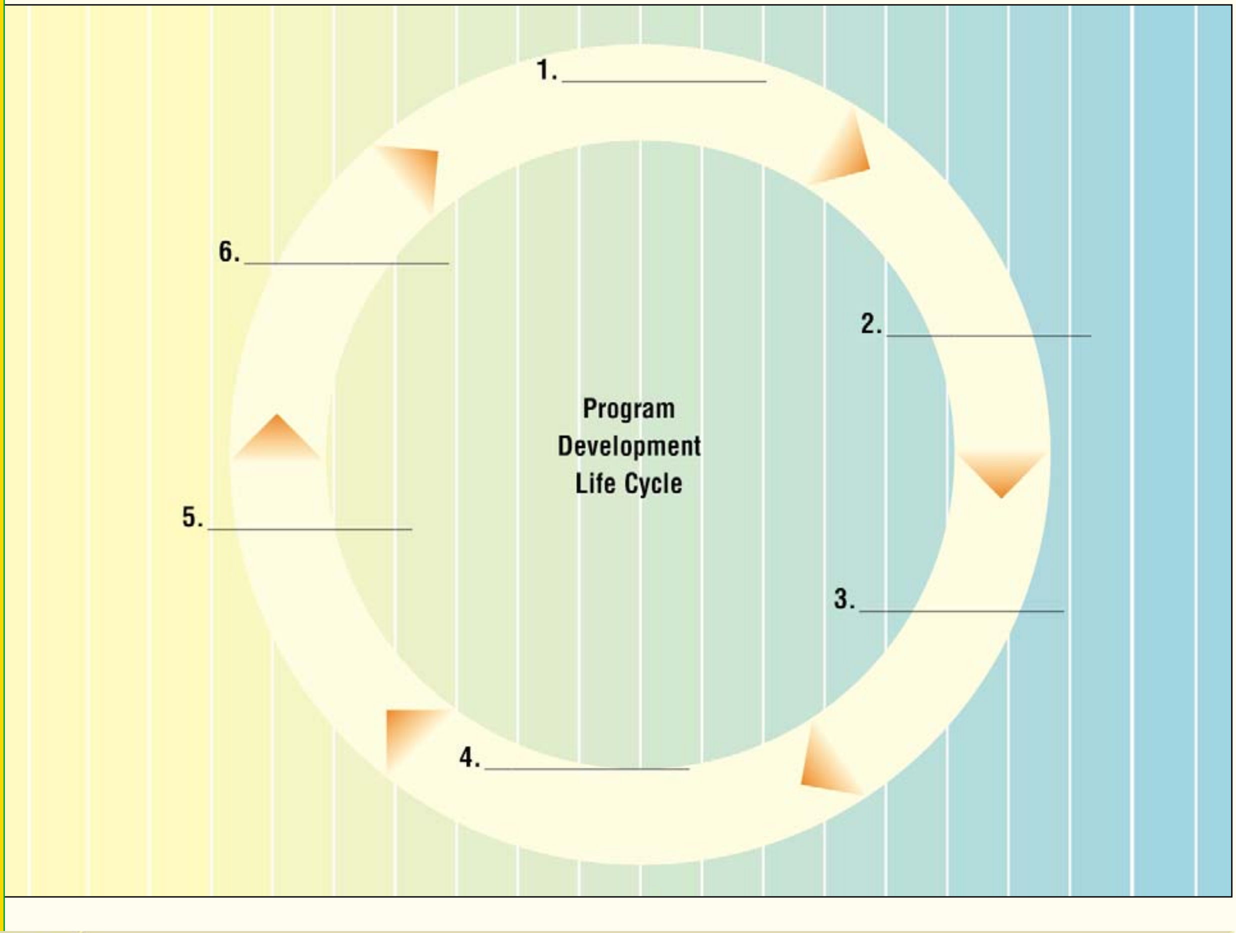
Click Practice Test. Answer each question. When completed, enter your name and click the Grade Test button to submit the quiz for grading. Make a note of any missed questions. If required, print a copy to submit to your instructor.

Checkpoint

Web Instructions: To display this page from the Web, start your browser and enter the URL scsite.com/dc2002/ch15/check.htm. Click the links for current and additional information. To experience the animation and interactivity, Shockwave and Flash Player must be installed on your computer (download by clicking here.)

LABEL THE FIGURE

Instructions: Identify the steps in the program development lifecycle (PDLC).



MATCHING

Instructions: Match each term from the column on the left with the best description from the column on the right.

_______ 1. annotation
_______ 2. terminal
_______ 3. process
_______ 4. decision
_______ 5. input/output

a. Program instructions that transform input into output.
b. Beginning or ending of program.
c. Entry from or exit to another part of flowchart on the same page.
d. Entry from or exit to another part of flowchart on a different page.
e. Condition that determines a specified path to follow.
f. Enter date or display information.
g. Additional descriptive information about the program.

Checkpoint

MULTIPLE CHOICE

Instructions: Select the letter of the correct answer for each of the following questions.

1. A tool the programmer uses during top-down design where the program modules are drawn as rectangles is a(n) ___________ .
   a. IPO chart
   b. hierarchy chart
   c. Nassi-Schneiderman chart
   d. program flowchart
2. A(n) ___________ structure shows one or more actions following each other in order.
   a. sequence control
   b. selection control
   c. case control
   d. if-then-else
3. A Windows-based application programming language is ___________ .
   a. COBOL
   b. BASIC
   c. Visual Basic
   d. C
4. With a(n) ___________ , you can build an application without writing extensive programming code.
   a. Nassi-Schneiderman chart
   b. Delphi
   c. HTML
   d. application generator
5. ___________ is a compiled object-oriented programming language used to write applets.
   a. C
   b. RPG
   c. JavaScript
   d. Java

SHORT ANSWER

Instructions: Write a brief answer to each of the following questions.

1. How is the if-then-else control structure different from the case control structure? ___________ How is the do-while control structure different from the do-until control structure? ___________
2. What is proper program design? ___________ What are dead code, infinite loops, entry points, and exit points? ___________
3. How is a compiler different from an interpreter? ___________ What is an advantage of an interpreter? ___________
4. What factors should be considered when selecting a programming language? ___________
5. What is a CGI script? ___________ What is the difference between Jscript and JavaScript? ___________ What is an applet? ___________

WORKING TOGETHER

Instructions: Working with a group of your classmates, complete the following team exercise.

If you work in an office and use a computer, you probably have experienced an out-of-date computer or last year's software. To address this problem, software industry predictions indicate we are moving into the next wave of the Web where software and services are delivered online. Microsoft's Office Online, for example, provides a choice for Office customers who prefer the benefits of centrally managed software. Investigate the possibilities of online software. Create a three- to five-page report, addressing the positives and negatives of offering software online. In particular, focus your report on software errors and how those errors could impact this offering. To learn more about software online, visit scsite.com/dc2002/ch15/check.htm.

In The Lab

Web Instructions: To display this page from the Web, start your browser and enter the URL scsite.com/dc2002/ch15/lab.htm. Click the links or current and additional information.

Searching for Executable Files

1. This exercise uses Windows 98 procedures. In Windows, application files have an .exe extension, meaning that they are executable. To find the executable program files on your computer, click the Start button on the taskbar, point to Find on the Start menu, and then click Files or Folders on the Find submenu. In the Find: All Files window, click the Name & Location tab, click View on the menu bar, and then click Details. Type `*.exe` in the Named text box. Type `c:\` in the Look in text box. Make certain the Include subfolders check box is selected. Click the Find Now button. Sort the files alphabetically by name by clicking the Name column heading. Scroll through the list of files and find the file name, Notepad. In what folder is Notepad located? How large is the file? Close the Find window.

Movie Box Office Simulation

2. Insert the Discover Data Disk into drive A. See the inside back cover of this book for instructions for downloading the Discover Data Disk or see your instructor for information on accessing the files required in this book. Click the Start button on the taskbar and then click Run on the Start menu. In the Open text box, type `a:movie.exe` and then press the ENTER key. This program was written in Visual Basic. The first customer wishes to purchase three tickets to a matinee performance of *The Abyss*. Notice the amount due and then click the Enter button. Enter the following transactions and write down the amount due for each transaction: (1) *The Client*, no matinee, 3 tickets; (2) *Forrest Gump*, matinee, 1 ticket; (3) *Beverly Hills Cop*, matinee, 2 tickets.

Adjusting Keyboard Speed

3. Windows allows users to adjust the keyboard to their own specifications. To customize the keyboard, click the Start button on the taskbar, point to Settings on the Start menu, and then click Control Panel on the Settings submenu. Double-click the Keyboard icon in the Control Panel window. When the Keyboard Properties dialog box displays, if necessary, click the Speed tab (Figure 15-45). Use the Question Mark button on the title bar to answer the following questions: What is Repeat delay? What is Repeat rate? Click the Cancel button. Close the Control Panel window.

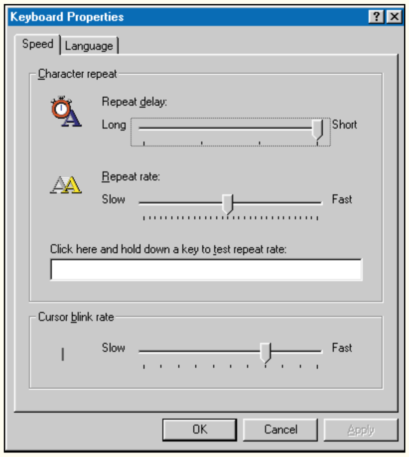


Figure 15-45

Loan Payment Calculator

4. Insert the Discover Data Disk into drive A. See the Preface at the front of this book for instructions for downloading the Discover Data Disk or see your instructor for information on accessing the files required in this book. Click the Start button on the taskbar, and then click Run on the Start menu to display the Run dialog box. In the Open text box, type `a:loancalc.exe` and then press the ENTER key to open the Loan Payment Calculator window. This program was written in Visual Basic. Type `12500` in the LOAN AMOUNT text box. Click the YEARS right scroll arrow or drag the scroll box until YEARS equals 15. Click the APR right scroll arrow or drag the scroll box until APR equals 8.5. Click the Calculate button. Write down the monthly payment and sum of payments. Click the Clear button. What are the monthly payment and sum of payments for each of these loan amounts, years, and APRs? (1) 28000, 5, 7.25; (2) 98750, 30, 9; (3) 6000, 3, 8.75; and (4) 62500, 15, 9.25. Close the Loan Payment Calculator.

Web Work

Web Instructions: To display this page from the Web, start your browser and enter the URL scsite.com/dc2002/ch15/web.htm. To view At The Movies in exercise 1, RealPlayer must be installed on your computer (download by clicking here). To use the Shelly Cashman Series Choosing a Programming Language Lab from the Web, Shockwave and Flash Player must be installed on your computer (download by clicking here).

Java Victory

To view the Java Victory movie, click the button to the left or click the Play button to the right. Watch the movie, and then complete the exercise by answering the questions below. Developed by Sun Microsystems, Java is an important language used in Web site development, particularly in the creation of graphic elements such as moving banners, ads, and logos. Java was designed by Sun as an easy-to-use, cross-platform language that would run on any computer. Sun licensed Java to others with the hope that Java would become the standard language of the future. Microsoft, an early licensee, chose to modify its version of Java only to run on Microsoft operating systems. Sun sued, and subsequently won the case. What are the benefits of standardization? Do consumers or companies benefit more?

Shelly Cashman Series Choosing a Programming Language Lab

Follow the instructions in Web Work 2 on page 1.47 to start and use the Choosing a Programming Language Lab. If you are running from the Web, enter the URL, www.scsite.com/sclabs/menu.htm; or display the Web Work page (see instructions at the top of this page) and then click the button to the left.

Image Maps

One common use of a script is to create an image map — a Web page picture that points to a URL and is used in place of, or in addition to, plain text hyperlinks (Figure 15-46). When you click a part of the picture, the Web browser locates the corresponding URL and sends the Web page to you computer. To work with image maps on Web pages, click the button to the left and complete this exercise.

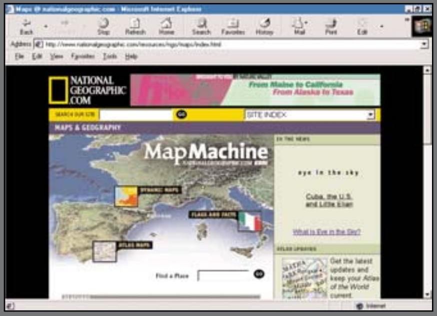


Figure 15-46

Application Generators

When using an application generator, a developer works with menu-driven tools that have easy-to-use graphical interfaces. To use an application generator to produce Web pages, click the button to the left and complete this exercise.

In the News

Programming is not just for programmers anymore. Yaroze is a Japanese expression meaning, let us work together. Net Yaroze is a new project that lets anyone with a computer and some programming experience work together with Sony to create their own games for Sony's Playstation. Increasingly, entertainment and productivity applications are offering tools, such as macros, that allow users to program their own innovations. Click the button to the left and read a news article about programming. Who is doing the programming? How is the programming different?